University of California, Berkeley

Electronics for the Internet of Things

ME 100 Course Notes

Massimiliano de Sa

These notes were written for the UC Berkeley course ME 100, Electronics for the Internet of Things in the Fall 2021 semester. I hope you find them useful!

Contents

1 Programming with Python

In this section, we'll provide you with a brief overview of the Python Programming Language, the language used for the labs in this course. If you're a mechanical engineer, it's likely you've come from a course like E7 (Intro to Computer Programming for Scientists and Engineers), where you learned to code in MATLAB.
Just like MATLAB, Python is a programming language where you can code anything you want! Why use it? Python is one of the most popular programming languages out there, and is gaining popularity in almost every field. A few of the reasons for this are:

1. **It's easy to read and write!** Whereas other programming languages can often be cumbersome to write in, Python is a very "readable" programming language.

2. **It's open source.** Unlike MATLAB, which is licensed and distributed by the company MathWorks for a fee, Python is entirely free.

3. **It's widely used.** According to Northeastern University, Python is currently one of the most popular programming languages in the world.

In ME 100, you'll become familiar with Python, and will come to appreciate the elegance and power of the language.
In this chapter, we'll review some of basic programming concepts and strategies, and show you side by side comparisons of programs in MATLAB and their equivalents in Python to get you up to speed in the language.

1.1 Programming Preliminaries

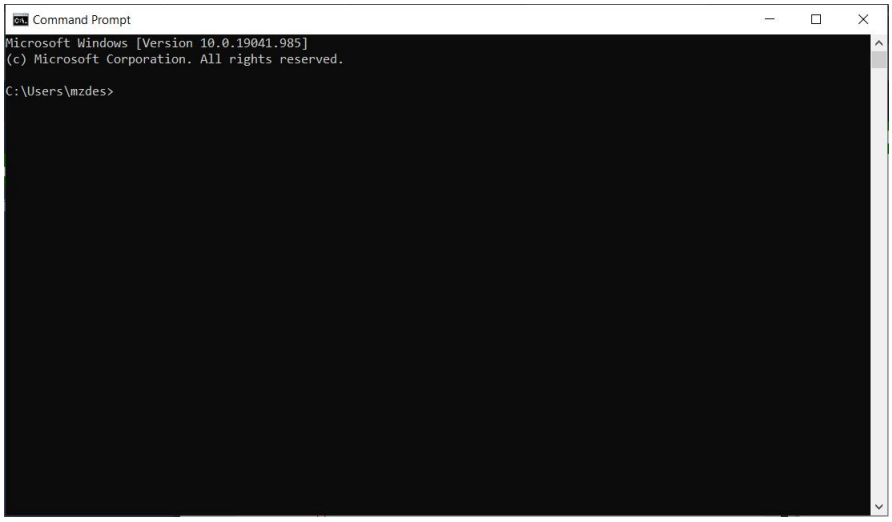
1.1.1 The Command Line Interface

As you may recall, in MATLAB, after you've finished writing your program you use the "command window" to run it. With Python, and many other programming languages, you use a similar but much more powerful command line, one that has access to a variety of computer functions.

Above: The MATLAB command window (found at the bottom of every MATLAB window)

Before we get started writing programs in Python, it's important to learn your way around your computer's command line, or 'terminal,' interface.

By typing "command prompt" on Windows or "terminal" on Mac into your search bar, you'll open a window that looks similar to the following:



Above: The Windows command prompt (note that your computer username will appear after 'Users')

All the way to the left of your command prompt, you'll find a line that says 'C:\Users\username>'. This is the **directory** that your computer is currently in. You can think of the directory as your "location" in the computer.
Although this may seem like new terminology, you've been using directories ever since you first turned your computer on! Every folder when you open up your file explorer - from desktop, to documents, downloads, and more - is a directory. These are all locations in your computer.
How do we move to a different directory? The "**cd**"[1] command (**c**hange **d**irectory) may be typed into the command prompt to help us navigate through our computer's file system.
As an example, let's see how we change to the desktop directory from our starting, or "root" directory. Type in the command "cd Desktop" to change to the desktop directory.

```
C:\Users\mzdes>cd Desktop

C:\Users\mzdes\Desktop>
```

As you can see by the second line, we've now successfully navigated to the desktop directory.
Now that we're in our desktop directory, we might want to create a new directory (folder) to store our ME 100 class files in. Instead of doing this from the file

[1] Computer commands will be bolded in this chapter

explorer, let's try it from the command line!
To make a new directory, type "**mkdir name**", where "name" is the name you want for your new folder.

```
C:\Users\mzdes\Desktop>mkdir me100
```

After hitting enter, you'll have created your new ME 100 directory. Congratulations! Try navigating to your new directory by using the **cd** command.
Now that we've made and entered our new directory, what happens if we want to go back? To go back by one directory, type the command "**cd ..**". To go all the way back to your root directory, simply type "**cd**".
When working on the command line, it's often easy to get lost among all the commands and folder names. To help navigate the command line, and to help avoid getting lost in your system's vast storage, the command "**dir**" (**ls** for Mac users) helps you find where you are. Typing this command gives you a list of all the files and folders in your current directory.
The following table outlines a set of useful command line commands:

Action	*Command*
Change Directory	$cd < name >$
Move Back	$cd ..$
List files	*dir* ('*ls*' *for Mac*)
Make a new directory	$mkdir < name >$

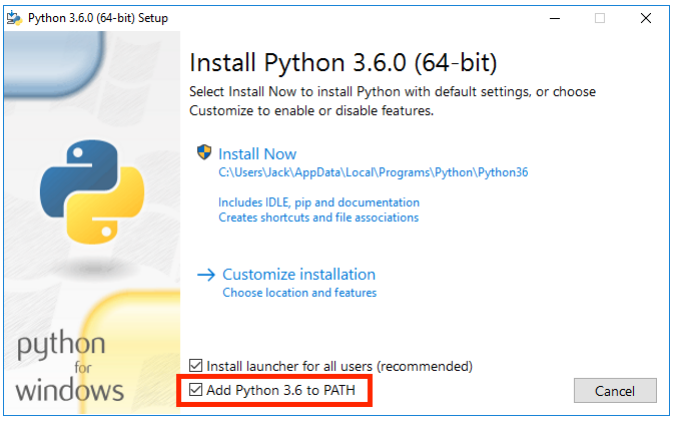
Let's get started with some Python!

1.1.2 Installing Python

Before you begin programming, it's important to make sure you have Python installed on your computer. If you're on Mac or Linux, it's likely you already have Python installed, but if you're on Windows and have never used Python before, you'll need to complete a fresh install.
Follow the following steps to get Python up and running on your PC:

1. Go to the Python website. Under the downloads tab, find "all releases," and click the Download for Windows option.

2. After downloading the Python installer, click on it to run. Go through the setup, following the default options. **Make sure** to check off "**Add Python 3.X to PATH**." This is an important step for making sure Python is easily accessible from the command line.



3. Finish the steps in the installer. Python will now be installed!

1.1.3 Basic Python Commands

Now that we know the basics of the command line, we can introduce the Python Programming language.
As you may recall, when writing MATLAB, you had to open up an application and hit "run" to execute your programs. With Python, the process is slightly different!
In Python, we can run our programs and write Python code directly from our computer command line. Type '**python**' (**python3** for Mac users) into your terminal and hit 'enter' to get started. Your terminal should now look something like this:

```
C:\Users\mzdes>python
Python 3.8.5 (default, Sep  3 2020, 21:29:08) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32

Warning:
This Python interpreter is in a conda environment, but the environment has
not been activated.  Libraries may fail to load.  To activate this environment
please see https://conda.io/activation

Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Above: Note - Don't worry if you get a warning - as long as you see the >>>, you're ready to get programming!

Whenever you see '>>>' in your terminal window, know that whatever command you type will now be interpreted by the computer as a Python command. Command line commands such as **cd** or **mkdir** will thus no longer be understood!
Let's start with a basic print statement. To print the text "ME 100" in Python, type "**print('ME 100')**". This is comparable to writing "disp('ME 100')" in MATLAB.
Let's look at what happens when we use the print command in the terminal window:

```
>>> print("hello world")
hello world
>>>
```

As you can see, the message in quotes is printed to the terminal in the line below the command. Note that in Python, it doesn't matter whether you use single or double quotes around strings - just stay consistent with your choice!
Just as we may use the print function in the terminal, we can use any other function built into Python. Below, we present the notation for basic arithmetic. Note that exponents in python don't use the carrot symbol from MATLAB, but rather use ** instead.

```
>>> 2+2
4
>>> 2*2
4
>>> 2/2
1.0
>>> 2-2
0
>>> 2**2
4
```

In the table below, you'll find the basic math operations in Python:

Operation	*Command*
Addition	+
Subtraction	−
Multiplication	*
Exponentiation	**
Division	/
Floor division	//
Modulo (remainder)	%

Note that in Python, unlike in MATLAB, we don't include a ';' at the end of every line. Python instead automatically recognizes each new line as a new command.
To exit Python and return to your normal command line, simply type **exit()** into the terminal window.

1.1.4 Variables and Data Types

In every programming language, we use *variables* to store information for future use. In Python, the way in which we define and use variables is much the same. Variables are assigned values using the = sign. In the image below, you'll find variables of several important data types.

```
>>> x = 2
>>> y = 'hello there'
>>> z = True
>>> a = 2.0
>>> b = [1,2,3,4,'c','x','z']
>>> c = None
```

Let's review the details of some important Python data types.
Integers work slightly differently to MATLAB. In MATLAB, we commonly allocate the size allowed for an integer by writing 'int8' or 'int16', for example. In python, however, the size is automatically allocated, and there is only a single 'int' type.
Boolean values are either True or False. Note that writing out True/False is not the only way of writing boolean valued variables in Python. Other False values include but are not limited to None, 0, and the empty list [ ], while True values are everything else.
What does this mean? Python will actually evaluate things like the integer 1000, the string "ME 100" and more as True!

```
#examples of True Values:
True
1
10
'x'
'ME 100'
```

Floats are floating point decimals. Note that the output of a division operation will *always* be a float, no matter if the two numbers divide evenly or not.
Lists are 1-dimensional arrays of numbers. Denoted by square brackets [ ], their elements are separated by commas. Note that elements of a list *do not* have to have the same data type.
Just as in MATLAB, we may access the elements of a list by *indexing* into the list. Unlike MATLAB, however, Python (and most other programming languages) assigns the first element an index of 0, rather than 1. For the list **b** in the above image, for example, typing in b[0] will return 1, the first element. Typing in b[-1] conveniently returns the last element of the list, 'z'. len(b), for a list named b, will return the length of the list. Unlike MATLAB, Python *does not* have built-in support for matrices.[2]
In the below example, we create another list, x[3]:

```
#define a list 'x'
x = [1, "ME 100", 3, True, 5]
x[0] #access and print the 1st element
>>> 1
x[-1]
>>> 5
```

[2]The NumPy library adds matrix operations to Python if you'd like to work with them in your code.
[3]Note that in code examples, ">>>" represents Python's output, *not* code that you have to write.

```
len(x)
>>> 5
```

Strings are sequences of characters. You may index a string to work with elements just as you would with a list. Strings may be joined together, or 'concatenated,' with the '+' operator.

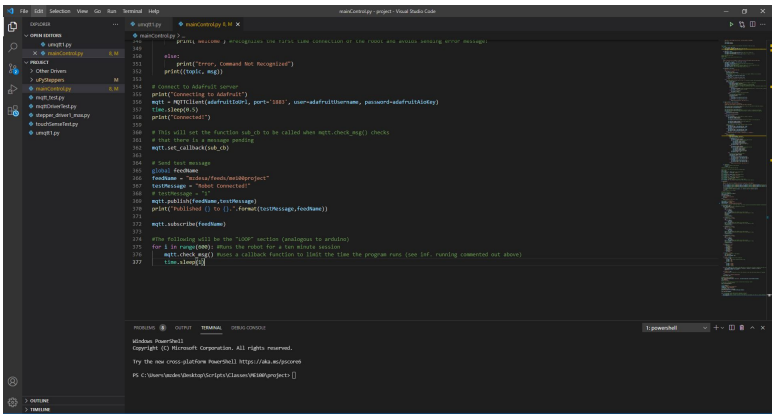
```
#define a string 'y'
y = 'hello'
y[0]
>>> 'h'
z = ' there'
y+z
>>> 'hello there'
```

1.1.5 Text Editors

Now that we've begun to write basic Python statements from the command line, you might be wondering how we can write larger Python programs and *save* our files?
To do this, we use what's called a **text editor**. A text editor is a code-writing application that lets us conveniently write, edit, save, and run code. Think of it like Microsoft Word, but for writing programs!
The following are some popular text editors.

1. **Visual Studio Code** (Has a *very* useful built-in command prompt feature, requires very little setup)
2. Sublime Text
3. Atom
4. Vim (Not recommended unless you have prior experience)

In the following sections, we'll be using Visual Studio Code (VS Code) to write our programs, but feel free to use the text editor you like the best!



Above: The VS Code interface. Notice the built-in terminal window at the base of the screen, which can be useful for quick debugging and running of programs.

Beyond simple text editors, more advanced editors called Integrated Development Environments (IDEs), similar to the MATLAB interface, are available. These provide extra support in a wide range of avenues, from debugging to testing and beyond. If you're interested in trying a popular Python IDE, PyCharm is a great place to start. Note that for the purposes of ME 100, however, it'll mostly be easier to use a normal text editor such as VS Code.

1.1.6 Using Visual Studio Code

In this section, we'll assume you've downloaded VS Code and are using it as your text editor for the course. Note that the topics discussed in this section are very transferable to other editors. Let's begin!
When you open VS Code, the first thing you should do is open the directory you'll be working in. Let's use our ME 100 directory from earlier.
To open this directory, go to file/open folder, and select the ME 100 folder you created in the command line tutorial (or any other directory you'd like to work in). Next, now that you're in your ME 100 folder, click file/new file.
When you create a file, it's good practice to save it right away! This is important so VS Code recognizes that your file is a Python file. Save your file as "**name.py**". The ".py" extension allows VS Code to recognize the language and make it easier for you to write in Python! Python code will now have colored highlights to help identify types of commands.
Next, we'll open up a built-in terminal inside VS Code. To open a built-in terminal, go to the menu at the top and select terminal/new terminal.
Now, we're ready to learn more about the structure of Python code.

1.2 Programming Concepts

1.2.1 If-Statements

When writing code, we often want our program to make decisions and execute pieces of code depending on whether certain conditions are met. We do this using an "if" statement.
To use an if statement in Python, we type "if", followed by a logical expression that evaluates to a True or False value (such as $2 > 1$ or $a <= b$), a colon, and then our block of code to be executed should the expression be true.

```
if expression: #if true
    code block #evaluate code
```

If we want to evaluate for multiple conditions, we use the following:

```
if expression P:
    code block 1
elif expression Q:
    code block 2
elif expression R:
    code block 3
else:
    code block 4
```

Above, we see that we use the command "**elif**" when we have multiple logical expressions. Note that the command "**else**" is optional, and does not have a logical statement - the "else" code block is executed if and only if none of the other expressions are true.
Below, you'll find the equivalent expression to the above in MATLAB.

```
if expression P
    code block 1
elseif expression Q
    code block 2
elseif expression R
    code block 3
else
    code block 4
end
```

Below, you'll find a table of common logical comparisons in Python:

Operation	*Python Command*
Greater than	>
Greater than or equal	>=
Less than	<
Less than or equal	<=
Equal	==
Not equal	!=
Logical and	*and*
Logical or	*or*
Logical not	*not*

Make a special note of the equal to logical expression (==). Make sure that when you're trying to compare two values, you use x==y, rather than x=y. A single equals sign denotes *assignment* (sets the value of x to be y), while a double equals sign denotes *comparison* (asks if x is equal to y).

1.2.2 Loops and Iteration

Iteration, the process of repeating a block of code, is a common procedure in programming. Similar to MATLAB, Python is able to achieve this with both while and for loops. Let's examine the syntax for Python loops.
Recall that in a *while* loop, the following procedure is evaluated:

1. Evaluate the header expression. If True, continue to step 2, if False, exit the while loop.
2. Evaluate the body within the while loop.
3. Go back to step 1.

Whereas in MATLAB, a while loop has the structure:

```
while expression
    code block
end
```

In Python, we would write:

```
while expression:
    code block
```

Note that as with MATLAB, in python, we may use the "break" keyword to stop the loop at any time in its execution.
Another way of achieving iteration is to use a for loop. A for loop can be particularly helpful when trying to repeat a process a certain number of times. It also provides us with a convenient, clean way to iterate over lists.
Let's look at the for loop's implementation in MATLAB and Python.
In MATLAB, to use a for loop to repeat a process n times, we would write (for some integer n):

```
for i = 1:n
    code block
end
```

While in Python, to repeat something n times, we would write:

```
for i in range(n):
    code block
```

You might be wondering what the "**range**" function is, as it differs from what you may have seen in MATLAB programming. If you enter a single argument (n), the range function starts at 0 (inclusive) and counts up to but *not* including n.
Note that this is not the only way of using a Python for loop, but is rather the one most similar to MATLAB. Check out the Python documentation for other useful ways to use a for loop!

1.2.3 Running your Python Code

Now that we're writing Python code in files, it's important to know how to *run* it!
To run a Python program from the command line, first check that you're in the correct directory! This is the directory where the file you wish to run is held. If you're not in the correct directory, Python won't know where to find your file!
Next, to run your program, write (use **python3** on Mac instead of **python**)

```
python fileName.py
```

in the command line. Just like that, Python will run your program!
What if you want to run your code, but still be able to work with the variables afterwards? You can run python in **interactive** mode to do this.
To run your Python file in interacive mode, type:

```
python -i fileName.py
```

Interactive mode is particularly useful when you want to use a function you've defined!

1.2.4 Functions

Now that we've explored the syntax for the basic Python commands, we might wish to organize our program better. When we wish to use a set of commands over and over again, instead of rewriting them every time, we can define a *function* that carries those commands out.
Function definition is where the syntax differs fairly significantly between Python and MATLAB, but don't worry - Python function definition is much more convenient and simple to use, and you'll pick it up in no time!
Let's quickly review the key parts of a function:

1. **The Header**: The function header defines the name of the function.
2. **The Arguments**: The arguments (also known as parameters) are the inputs to a function. They're typically listed as variables in parentheses in the function heading. When we call a function, we bind values to these parameters.
3. **The Return Statement**: This (optional) part of a function determines what information is sent back to the program after a function is called.

Let's look at an example: a function "adder" that adds 3 arguments: a, b, and c. In MATLAB, we would write:

```
function [output] = adder(a,b,c)
    output = a+b+c;
end
```

In Python, to define the same function, we would write:

```
def adder(a,b,c): #defines the function name and arguments
    return a+b+c #returns a+b+c
```

Let's break down what's happening here.
The **def** keyword tells Python that we're about to *define* a function with the name adder. After the name, we list our arguments in parentheses, and separate them by commas. After closing the parentheses, we write a colon (:) to tell Python the body of the function is about to begin.

```
#General function definition
def name(arg1,arg2,arg3): #defines the function name and arguments
    #function body
    code block
    return expression #returns the value of ‘‘expression"
```

Next, we type the **body** of the function, which must be indented from the def statement. The body may be any number of lines long, and may include any code you wish.
Somewhere in the body of the function, you must include a **return** statement. The return statement sends information back to the program outside of the function. Here, we return the sum of the three arguments.
Note that after the return statement is executed, Python exits the function! This means that nothing more in the function will be executed after the return

statement is evaluated.
Now that we've defined our function, we might want to call it! To call our adder function, we simply write the function name, followed by values for its arguments in parentheses. For example:[4]

```
def adder(a,b,c): #defines the function name and arguments
    return a+b+c #returns a+b+c
adder(1, 2, 3)
>>> 6
```

You may have noticed the symbol "#" appearing in many of our Python examples. This denotes the start of a comment - text that will be ignored by Python that's just there for the user! It's good practice to comment your code well so others can understand it better!

```
#this is a comment! Python does not execute it.
```

Now that we've defined what a function is in Python, we must ask ourselves the question, "How can we make it simpler?"
For short, one-line functions, we often don't want to take the trouble to write out a full def statement. To make one-line function definition concise, Python has a special type of function known as a **lambda**.
To define a lambda for the adder function, we would write:

```
adder = lambda a,b,c: a+b+c
```

Instead of using the def statement, we define the function like a variable, and use an = sign. We write "lambda" to signify our function is starting, and then write our arguments, which are followed by a colon. The return value is then whatever follows the colon. Adder may then be called in exactly same way as a normal function. For example:

```
adder(1,2,3)
>>> 6
```

would return 6 to the terminal.
Remember, lambdas are only to be used when functions have single-line bodies! Otherwise, you must use a def statement to define your function.

1.2.5 Recursion

Recursion is an interesting, elegant strategy for repeating a procedure. It's a programming technique that involves a function *calling itself.*
Although you likely won't need to use recursion in your programs in ME 100, it's a valuable programming technique to have mastered in Python.
Commonly, we use recursion to solve problems where our original problem may be broken down into a simpler version of itself.
Consider the factorial function. If we want to find 5!, for example, we know it's the same as finding $5 \cdot 4!$. Now, we only need to consider the smaller problem of finding 4!. We continue simplifying until we hit the **base case**, the simplest

[4] When we call adder(1, 2, 3) we would do this in interactive mode. Use python -i filename.py to be able to try out your function!

form of the probelem.
Recursion takes advantage of the idea of "solving a simpler version" to produce clean, readable solutions to often complex problems in programming.
Let's now define a recursive factorial function in Python! Begin by writing a normal function definition.

```
def fact(n):
    #Return the factorial of n
```

We always begin by writing the base case, the simplest form of the problem. In this case, as written above, our base case is when n = 1, since we know 1! = 1.

```
def fact(n):
    #Return the factorial of n
    if n == 1: #use a double equal sign to compare values!
        return 1
```

Following this, we write our recursive case. This is where we call the function on a simpler version of the problem, using the idea that $5! = 5 \cdot 4!$.

```
def fact(n):
    #Return the factorial of n
    if n == 1: #use a double equal sign to compare values!
        return 1
    else:
        return n*fact(n-1)
```

Great! Let's try a second example: the Fibonacci Sequence, which is defined by adding together the two previous terms of the sequence.
First, we must define the *base case(s)* of the function. These are the first two terms of the Fibonacci sequence, and are the terms that stop the function from recursing forever.

```
#function for computing the nth fibonacci number
def fibonacci(n):
    #base cases:
    if n == 1:
        return 1 #first base case
    elif n == 2:
        return 1 #second base case
```

This defines the first two terms of the Fibonacci sequence. Now that we have the base cases complete, we must write the recursive case, which adds the two previous terms of the sequence.

```
#function for computing the nth fibonacci number
def fibonacci(n):
    #base cases:
    if n == 1:
        return 1 #first base case
    elif n == 2:
        return 1 #second base case
    else:
        return fibonacci(n-1)+fibonacci(n-2) #add the two previous
```

This completes the recursive Fibonacci function. As you can see, the function is able to call itself within its body to evaluate the nth term in the Fibonacci sequence.

1.2.6 Object Oriented Programming

Object Oriented Programming (OOP) is a programming technique that helps us write and organize large programs. It's very different to what we've written so far, which has been primarily *functionally oriented*.
In OOP, our program is centered around defining "**objects**" (variables) and "**methods**" (functions) that act on those objects.
Whereas in our programming so far, our variables typically only had one value, such as "2" or "hello world", objects are a much more general and powerful type of variable. Objects may hold many different pieces of information. For a bank account, for example, an account object might hold the owner name, start date, and account balance.
Let's write an object oriented program for a bank account. By the end of the program, we want to have attributes that store the bank account's balance, it's owner, and its date of opening. We also want to have methods that change the account balance whenever deposits or withdrawals are made. Let's begin!
The first step in any object-oriented program is to define a class. A class is something that holds all of the attributes and methods that may be associated with an object. After defining a class, we'll be able to define *objects* of the class's type.
We define a new class with the following line:

```
#bank account class
class Account:
```

Notice that it's convention to capitalize the first letter of the class name.
When defining a class, the first thing we do after writing **class Name:** is write the special "__**init()**__" method. This is a method that *initializes* every **attribute** of a bank account object. Attributes are variables associated with an object.

```
#bank account class
class Account:
    #Note that __init__ uses TWO underscores on either side
    def __init(self,bal,nme,dte)__:
        self.balance = bal #balance, name, and date attributes
        self.name = nme
        self.date = dte
```

Let's break down the different components of the __init()__ method. In the parentheses, each argument represents a value for each of the object's attributes. In the body, we bind these values to the attributes of the object.
Init is automatically called when the object is created. For example, writing:

```
#define a bank account object
my_account = Account(100, "Prof Anwar", "01/01/2021")
```

calls __init()__ to define a new bank account object that holds $100, is registered under the name "Prof Anwar", and was started on January 1st, 2021.
Let's continue to examine the function. What does "self" mean, and why do we use dots?
In OOP, we access the attributes of objects using **dot notation**. We write the

object name, a dot, and then the name of the attribute we'd like to get. For example, to get the balance of the my_account object, we would write:

```
my_account.balance #get the balance of my_account
>>> 100 #100 is returned
```

When defining our class, we want to make sure that we write our class as *generally* as possible. We want to write our class in terms of a *general* object, rather than a specific instance like my_account.
This is what the "**self**" object is. When writing a class, we always write it in terms of the object "self", which you can think of as a placeholder object for your class.
Now that we've finished defining the attributes of our class, let's define a method. Note that methods are exactly the same as functions, only specific to a class.
Let's first define a method to deposit money in the account. We'll take in the arguments: "self" (our general object) and "money," (the amount we want to deposit), add it to the account balance, and print a statement confirming our new balance. Note that "self" must **always** be one of the arguments of a method.

```
#bank account class
class Account:
    def __init(self,bal,nme,dte)__:
        self.balance = bal
        self.name = nme
        self.date = dte

    def deposit(self, money): #Always include self as an argument!
        #method to deposit money to the account
        self.balance = self.balance+money #add money to the balance
        print("Your new balance is $", self.balance)
```

Note that this is a good example of a method without a return statement! All we want to do is modify an object, rather than sending anything back to the rest of the program.
Let's now create our withdrawal function. Let's use an if statement to account for the case where we don't have enough money!

```
#bank account class
class Account:
    def __init(self,bal,nme,dte)__:
        self.balance = bal
        self.name = nme
        self.date = dte

    def deposit(self, money):
        #method to deposit money to the account
        self.balance = self.balance+money #add money to the balance
        print("Your new balance is $", self.balance)

    def withdraw(self, money):
        #method to withdraw from the account
        if self.balance-money<0: #if we try to withdraw too much
            print("You can't withdraw that amount.")
        else:
```

```
18              self.balance = self.balance-money #take money out
19              print("Your new balance is $", self.balance)
```

This completes the definition of our class. It now has all the functionality we might want a simple bank account to have.
Let's now see how we can use the Account class. First, we define our Account object. Then, we follow this by making a deposit of $100.

```
1  #bank account class
2  class Account:
3      def __init(self,bal,nme,dte)__:
4          self.balance = bal
5          self.name = nme
6          self.date = dte
7
8      def deposit(self, money):
9          #method to deposit money to the account
10         self.balance = self.balance+money #add money to the balance
11         print("Your new balance is $", self.balance)
12
13     def withdraw(self, money):
14         #method to withdraw from the account
15         if self.balance-money<0: #if we try to withdraw too much
16             print("You can't withdraw that amount.")
17         else:
18             self.balance = self.balance-money #take money out
19             print("Your new balance is $", self.balance)
20
21 my_account = Account(100, "Max", "01/01/2021")
22 my_account.deposit(100) #deposit $100 to the account
23
24 >>> Your new balance is $200 #print statement output
```

Note that just as with attributes, we also use dot notation when calling methods on objects. The format "**object_name.method_name(arguments)**" is always used in OOP. Note that the self argument is automatically filled in by Python, and we don't need to include it ourselves.
Why would we want to use this style of programming? As programs become larger, being able to organize variables using classes and objects can be very valuable for keeping a coherent, easy to read program. OOP is also very useful when we want to keep certain pieces of information together, such as balance, name, and incept date all being under the umbrella of a "bank account."
In ME 100, for example, you might want to create a class for a motor. You can include its input ports, its angular position, and more all in one motor object. As the semester goes on, try to recognize the features of OOP in the libraries you use in your labs!

1.2.7 Useful Libraries and Links

One powerful feature of the Python language is its ability to import groups of user-made functions, or "libraries," to increase the functionality of the language. For example, if you'd like to do math operations such as $\sin(x)$ or $\tan(x)$, which aren't built into basic Python, you can import the *math* library.

Below, you'll find a list of useful libraries and links that can be helpful to have both for general Python programming and for MicroPython[5] programming with electronics.

1. **NumPy:** NumPy is used to work with matrices and a variety of other mathematical constructs in Python.
2. **SciPy:** SciPy contains many useful scientific programming functions. For example, the function *odeint* provides a fast and reliable way of solving differential equations similar to MATLAB's *ode45* function.
3. **Matplotlib:** Matplotlib gives Python the ability to create graphs and visuals from data. It works similarly to the plotting functions you may be familiar with from MATLAB.
4. **Math**: The math library gives you access to commonly used trig, exponential, and log functions.
5. MicroPython Documentation: This link provides you with an overview of the functions built in to your ESP32 microcontroller. Check it out for a reference of how to interact with pins and use advanced functionalities like callback functions, PWM, and more!

Now that we know a few of the useful Python libraries, let's take a quick look at how we can import them into your programs. Note that we *always* import libraries at the top of our program so that all the lines of code below may access what's in the libraries.
The first way of importing libraries is to use the *import* keyword, as seen below.

```
import scipy
import numpy as np #imports NumPy with the nickname np
```

We can write "**import name**" if we want to use the library with its normal name, or "**import name as nickname**" if we want to work with a shorthand name for the library in our code. Note that numpy is commonly abbreviated as np.
Once we've imported libraries with this method, we use dot notation to access the functions within. For example, to create a numpy array after importing numpy as np, we could write:

```
numpy_array = np.array([1,2,3])
```

Note that we *must* use "np" here, otherwise Python will not know where to find the NumPy array function.
What if we don't want to use "np" every time we want to call a function from the NumPy library? Instead, using the *from* keyword, we can select particular functions to import directly into the program so we don't have to use the name of the library. Let's take an example from the math library.

[5] MicroPython is a version of Python that microprocessors such as the ESP32 can run

```
from math import sin
x=sin(10) #don't need to use math.sin
```

Note that above, since we used "**from library import function**", we don't need to specify what library we're taking the sine function from. In fact, we may use the from keyword to import the entire library as follows:

```
from math import *
```

The * tells Python to import *every* function from the math library. Now, we're able to use every function from math without using math.function().
Since this seems much more convenient, why would we not always use import all? One reason for this is that we don't know the names of everything that we're importing - that is, if there's a function in the library we're importing that happens to have the same name as a function we've defined in our program, we might have a conflict in our code that can lead to confusing errors.
As such, to stay organized, it's best just to import what you need.

1.3 Conclusions

If you're feeling overwhelmed, don't worry! Learning a new language is a daunting task! Over the course of the semester, you'll become more and more comfortable with Python.
If you're looking for some more practice, check out Coding Bat for some problems involving basic Python syntax. If you're interested in more in-depth, challenging projects, feel free to look at CS 61A, another great Berkeley course!

2 Introduction to Circuit Analysis

Circuit analysis is at the heart of hundreds of different technologies in engineering! From the ways in which our computers function to the methods we use to drive electric vehicles - everything is founded upon the basic principles of circuit analysis.
In this section, you'll learn the fundamental tools for analyzing linear resistive circuits, a class of circuit that's a stepping stone into the wider world of electronics!
Before we get into the details, let's do a quick review of some terms you may have seen before.

2.1 Basic Electrical Quantities

We may begin our exploration of circuits with a definition of some of the basic quantities we'll be working with.
Charge (Q) is a fundamental physical quantity held by protons and electrons. Electrons have a negative charge, while protons have a positive charge. As you may remember, opposite charges attract each other and like charges repel.
Current (I): Just as in thermodynamics, where we discuss the "mass flow rate" through a system, in electronics, it's often useful to think about the "charge flow rate."
Current is the rate at which charge flows through a conductor (often a metal wire) in a circuit. Current has units of amperes, or "amps." If a $1A$ current flows through a circuit, then 1 coulomb of charge passes through the circuit every second.
As current is the time rate of change of charge, we may express it mathematically with the derivative:

$$I = \frac{dQ}{dt}$$

For Q = charge and t = time.
Resistance (R), measured in Ohms, is an object's tendency to resist the flow of charge. For any wire or conductor, resistance is a function of three things: the length of the conductor, its cross sectional area, and a material property known as resistivity.
The following formula expresses this relationship:

$$R = \frac{\rho L}{A}$$

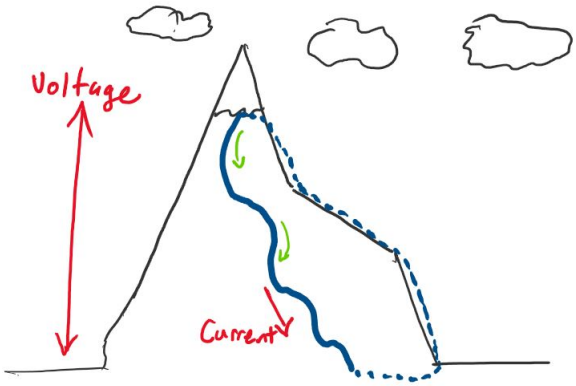
For ρ the resistivity, L the length of the conductor, and A the cross sectional area.
Voltage (V), measured in Volts, is defined as the potential energy per unit of charge between any two points in a circuit. As such, the voltage of a point is often referred to as the point's "**potential**." Voltage is always defined between two points, one of which is often a point known as the circuit "ground." This is a location in the circuit that's been assigned a potential of 0.

These four quantities are summarized in the table below.

Quantity	*Symbol*	*Units*
Charge	Q	*Coulombs* (C)
Current	I	*Amps* (A)
Resistance	R	*Ohms* (Ω)
Voltage	V	*Volts* (V)

2.2 Basic Circuit Elements

Now that we've defined some basic circuit *quantities*, we may learn how they're related by examining some basic circuit *elements*.
To develop an intuitive understanding of circuits, we often imagine an electrical circuit as a stream flowing down a mountain.
Current (I) may be thought of as the *speed* of the stream, while voltage is the stream's *elevation*, or *potential* on the mountain.



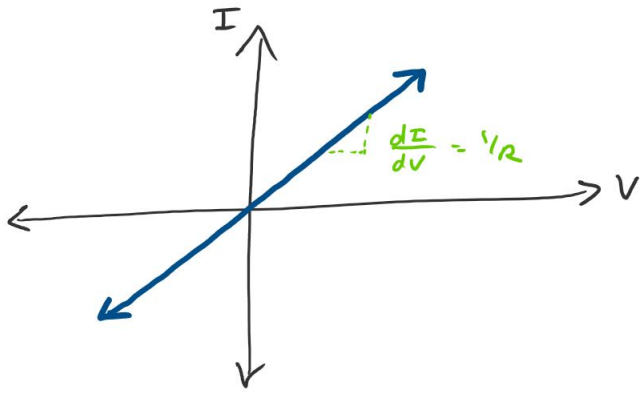
Above: The stream is pumped back up to the top of the mountain to complete the circuit

With this analogy in mind, we may define our first circuit element: the **resistor**.
A resistor *resists* the flow of current in a circuit. In our stream analogy, resistors would be the rocks, trees, and obstacles in the stream that slow the current down.
Because resistors limit current, we may say that they *dissipate* electrical energy. This causes a drop in voltage across every resistor, which may be quantified by **Ohm's Law**.

$$\boxed{V_r = IR}$$

Ohm's law states that the drop in voltage across a resistor is the product of the current going through the resistor and its resistance.
Plotting this relationship on a graph, with voltage on the x-axis and current on

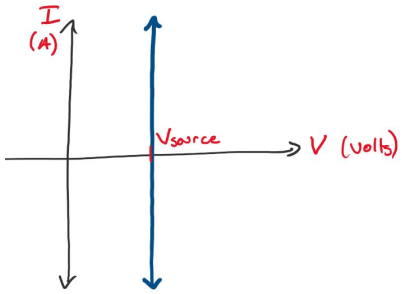
the y-axis, we observe the important fact that the relationship between current and voltage through a resistor is **linear**.



Although this mathematical representation of resistors is useful, oftentimes, it may help us to have a clear visual representation of our circuits as well.
To clearly express the layout of our circuits, we make drawings known as **circuit diagrams**. In these diagrams, resistors are drawn as follows:

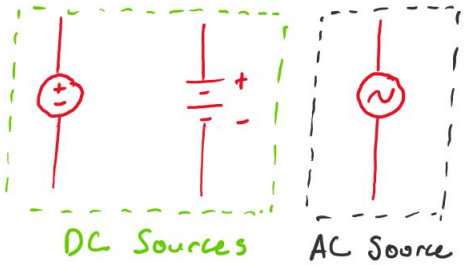
Above: A 10 Ohm resistor

Now that we've defined a component that *limits* the flow of current, we must define another component that *encourages* the flow of current.
The next circuit element we'll examine is the **voltage source**. In our mountain analogy, the voltage source is a pump that pushes the stream up to the elevation it needs to flow.
A voltage source has two terminals (positive and negative) that it *forces* a particular voltage across. No matter what else is happening in the circuit, that voltage will never change.
Because voltage sources aren't impacted by other parts of the circuit, they may have *any* current flowing through them. Thus, a voltage source may be represented by the below Current-Voltage graph:



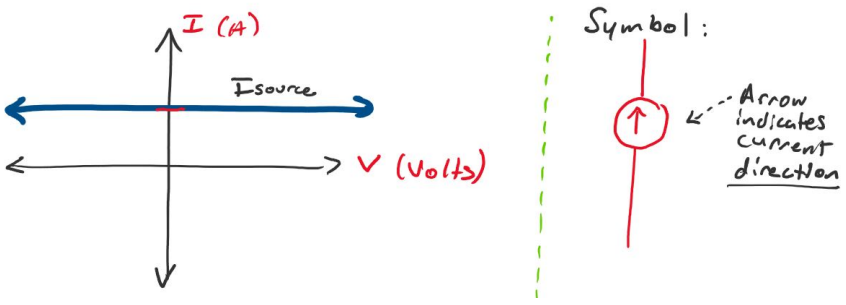
Above: A voltage source always stays at V_{source}, but may have any current running through it.

In circuit diagrams, voltage sources are typically depicted as one of the below:



Above: Two common symbols for DC sources are on the left, while the symbol for an AC source is on the right.

For now, we'll focus our studies on constant voltage sources (DC), whereas later, we'll examine time-varying sources (AC).
Logically, if there are voltage sources for circuits, there should be current sources as well!
Current sources are another important circuit component. Just as a voltage source forces a particular *voltage* across itself, a current source forces a particular *current* through itself.
In the mountain-river analogy, a current source is a pump that forces the stream to flow with a certain velocity.
A current source will always maintain its current no matter what else is happening in the circuit, and thus may have any voltage applied across it without any effects. Notice the interesting similarities in the properties of current and voltage sources!
Below, observe the symbol and current-voltage relationship for a current source.



The last fundamental circuit element we'll discuss in this section is the **wire**. As you may have guessed, wires are the electrical conductors used to connect the different circuit elements.

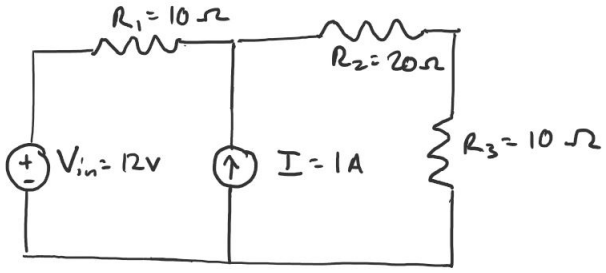
In our analyses, we assume wires to be ideal unless told otherwise. This means that they have zero resistance and may carry any current.

In reality, wires do have some internal resistance which we may occasionally need to account for, but for most analyses zero resistance is a valid assumption to make.

Let's think further about what it means to have zero resistance. By Ohm's law, $V = IR$, we find that no matter the current, there will never be any drop in voltage across an ideal wire, since $R = 0$. Additionally, if we apply any voltage across the ends of an ideal wire, the current will tend to infinity.

In circuit diagrams, wires are represented by simple straight lines.

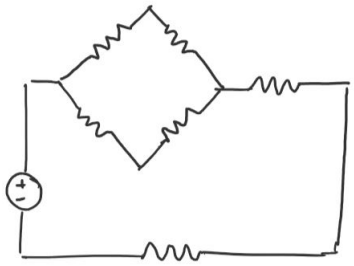
Putting all of the circuit elements together in a circuit diagram, we may now draw and understand circuits such as the following:



Above: an example of a "resistive" circuit. Try to identify the different circuit elements!

2.3 Equivalent Resistance

Oftentimes in circuit analysis, we're confronted with a confusing circuit diagram littered with resistors and other components. How can we simplify these complex resistive circuits to simpler yet equivalent forms?



Above: A resistive circuit we might want to simplify.

Using the idea of "equivalent resistance," we can simplify complex networks of resistors into a single **equivalent** resistor.
The following are layouts of resistors that may be easily simplified.

2.3.1 Series Resistance

When resistors are in **series**, they are connected end to end, as seen in the below drawing.

Above: Three resistors in series.

Let's develop an equation to simplify n series resistors of resistance R_i to one equivalent resistor with resistance R_{eq}.
Imagine that we apply a voltage V across a set of n resistors in series. Since all the resistors are connected end to end, the same current must flow through all n resistors.
Using Ohm's law, we know that the voltage across R_1 is $V_1 = R_1 i$, the voltage across R_2 is $V_2 = R_2 i$, and so on. We also know that the sum of the voltages must add up to the total voltage V. Mathematically, we express this as:

$$V = V_1 + V_2 + ... + V_n$$
$$V = R_1 i + R_2 i + ... + R_n i$$

Now, let's factor out current i from this expression. We then find:

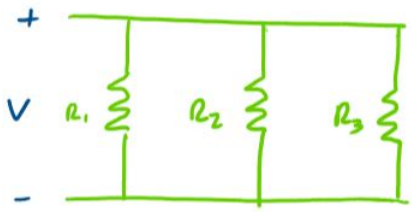
$$V = (R_1 + R_2 + ... + R_n)i$$

We now recognize the above as an expression of Ohm's Law, only now with $R = R_1 + R_2 + ... + R_n$. Thus, for n resistors in series, we define equivalent resistance as:

$$R_{eq} = \sum_{i=1}^{n} R_i = R_1 + R_2 + ... + R_n$$

2.3.2 Parallel Resistance

Parallel is a second common layout for resistors. When resistors are in parallel, the voltage across each element is the same. Parallel resistors might appear as follows:



Let's now determine a formula for the equivalent resistance of n parallel resistors. Imagine applying a voltage V across all of the resistors, as in the graphic above. Although the voltage is the same across each resistor, current is not necessarily the same, since each resistor lives on a separate path. Thus, we may write:

$$\begin{aligned} i_1 &= V/R_1 \\ i_2 &= V/R_2 \\ &... \\ i_n &= V/R_n \end{aligned}$$

Logically, the total current going through all of the parallel resistors is equal to the *sum* of the currents going through each resistor.[6] Therefore:

$$\begin{aligned} i &= i_1 + i_2 + ... + i_n \\ i &= V/R_1 + V/R_2 + ... + V/R_n \end{aligned}$$

Now, we may factor out V to find:

$$i = (1/R_1 + 1/R_2 + ... + 1/R_n)V$$

Once again, we recognize this as an expression of Ohm's Law: $i = V/R$. Thus, the equivalent resistance for n resistors in parallel is:

$$1/R_{eq} = (1/R_1 + 1/R_2 + ... + 1/R_n)$$

2.4 Power

In electrical circuits, whenever we pass a current through a resistor, the resistor will dissipate electrical energy in the form of heat. If you've used a laptop, you've likely felt the results of this firsthand!
The rate at which electrical energy is dissipated is known as **electrical power**. How can we quantify the power being dissipated by a component? We can start

[6] We'll develop this idea formally using Kirchoff's Current Law

by recalling an interesting fact from electromagnetism. When a charge Q is moved through a field with constant potential V, the magnitude of **work** done on the charge is:

$$W = QV$$

Now, recalling that the time derivative of work is power, we take the derivative of both sides of this equation to obtain (For constant V):

$$P = \dot{Q}V + Q\dot{V}$$
$$P = \dot{Q}V$$

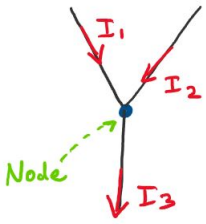
This equation may now be simplified further! Recalling that $I = dQ/dt$, we define the power dissipated by a component as:

$$P = IV$$

For I the current through the component and V the voltage across the component.
In electronics, we define a sign convention for power. If a device *supplies* power, P is negative. If a device *absorbs* power, P is positive. For example, as a resistor absorbs power, the power dissipated by a resistor is positive.

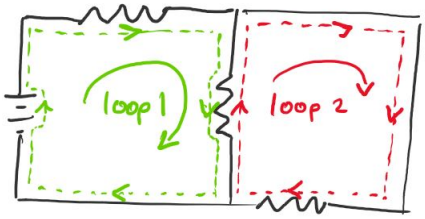
2.5 Kirchoff's Laws for Circuit Analysis

Just as we apply Newton's laws to solve problems in mechanics, in electronics, we also apply a set of laws to solve problems with circuits.
One of the most useful set of laws for circuit analysis is **Kirchoff's Laws**. Before we get into the details of Kirchoff's Laws, it's important to define a few more terms.
The first of these terms is the **node**. A node is an intersection in a circuit where two or more circuit elements (wires, resistors, sources, etc.) meet. We see one such example below:



Above: A node with two entering currents and one exiting current.

The second term we'll define is the **loop**. A loop is any path in a circuit that starts and ends at the same point and doesn't intersect itself. Some examples of loops are found below:



Note that we may also define a loop around the *entire* above circuit. Also note that the directions of each loop, denoted by the arrows, are entirely arbitrary and may be chosen by you.
Now that we've defined these essential terms, we're ready to understand Kirchoff's Laws!

2.5.1 Kirchoff's Current Law

The first of Kirchoff's laws is **Kirchoff's Current Law**, or **KCL** for short. KCL states that the sum of currents flowing into or out of any node a circuit is zero. Before we express this mathematically, let's develop an intuitive understanding of this law.
Imagine you have a box of marbles. At one end of the box, you have a pipe where marbles can come in. At the other end of the box, you have a hole through which marbles can leave.
From conservation of mass, we know that marbles can't simply "pop into existence" inside the box - any change in the number of marbles must thus be from the pipe or hole.[7]
Since we're not adding any marbles to the box, we can express the rate of change of the number of marbles in the box through the following relationship (for i entering and e exiting):

$$\frac{dM}{dt} = \dot{M}_i - \dot{M}_e$$

Since from conservation of mass, we know that the total mass of marbles must stay the same, the total rate of change of mass must be zero. Therefore:

$$0 = \dot{M}_i - \dot{M}_e$$

Now, let's translate this example of marbles and boxes to circuits. We replace "boxes" with nodes, and "marbles" with charges. For any node in a circuit, we may then write:

$$0 = \dot{Q}_i - \dot{Q}_e$$

Now, we recall a useful fact about charge and its relationship with current. We know that the time derivative of charge, $\dot{Q}$, is equal to current, I. Thus, we may then write:

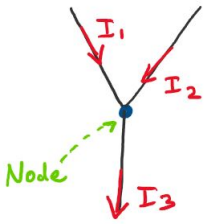
$$0 = I_i - I_e$$

[7] If you've taken ME 106, you'll recognize this as an application of control volume analysis

This is Kirchoff's Current Rule for a node with 1 entering wire and 1 exiting wire. Note that I_e is negative by convention.
Let's now generalize this formula to any number of entering and exiting currents. For a node with n number of currents, we can mathematically express KCL as:

$$\boxed{\sum_{i=1}^{n} \beta_i I_i = \beta_1 I_1 + \beta_2 I_2 + ... + \beta_n I_n = 0}$$

For I_i a current flowing into or out of a node and $\beta = \pm 1$ ($+1$ for current flowing in, -1 for current flowing out).



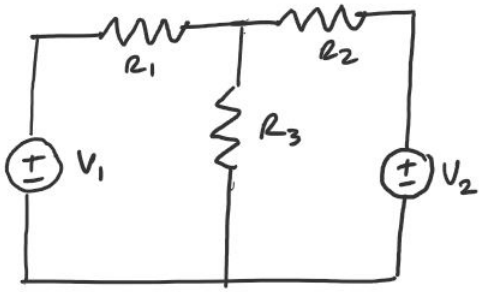
For the node above, for example, we would express KCL as:

$$I_1 + I_2 - I_3 = 0$$

In summary: Kirchoff's current law is a statement of *conservation of charge*. Whatever goes in must come out!

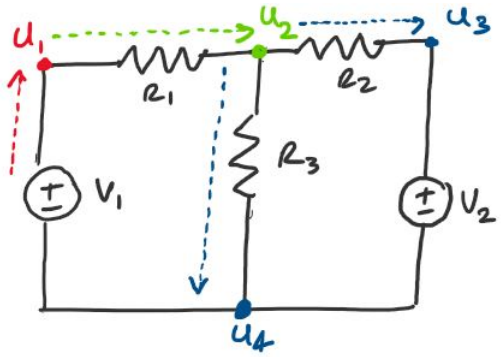
2.5.2 Applying KCL in Nodal Analysis

Now that we've developed an understanding of what KCL is, we may develop a step-by-step method for solving circuits using KCL. The following example illustrates a KCL technique known as "nodal analysis."
Ex: Solve for the currents and voltages across R1, R2, and R3



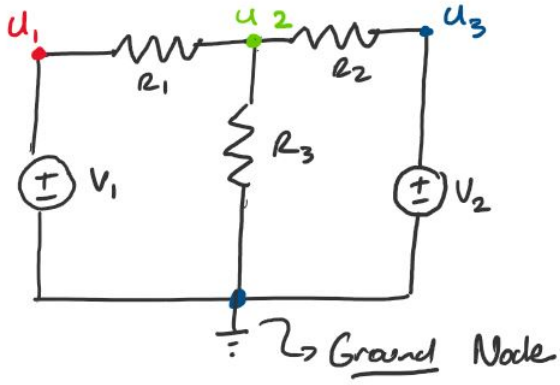
We follow the following procedure to "solve" a circuit (find all currents and voltages) using KCL:

1. **Identify all of the nodes in the circuit.** Recall: a node is a point where two or more circuit elements meet or where there is a junction in the wires. To identify all the nodes, it's good practice to start at the leftmost voltage source and trace the circuit path rightwards. Here, we begin at the voltage source V_1. We trace the circuit path until we meet the next element: R_1 - since two circuit elements meet, this point is a node. Continue tracing until all nodes have been labeled.



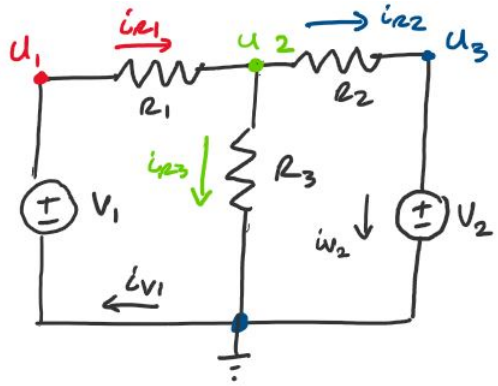
   Label each node u_i for i the number of the node. Each u_i represents the potential of that node.

2. **Identify a node as ground.** After labeling our nodes, we select one of the nodes to be the reference, or "ground," node. This node is automatically assigned a voltage potential of 0, and is what all other node potentials are defined relative to. By convention, we choose the last node in the circuit as ground (note that this is an arbitrary choice). Here, we choose u_4 to be ground.



3. **Assume current directions.** Label the current through each component as well as its direction (draw in with an arrow). For example, label the

current through resistor R_1 as I_{R_1}. For the direction, assume any direction - if you get a negative after solving, you know that current was simply traveling opposite to the direction you assumed.



4. **Write the KCL equation for each node.** Remember to apply the following convention for sign: current **in is (+)**, current **out is (-)**. In order of node number, we write:

$$I_{V1} - I_{R1} = 0 \tag{1}$$
$$I_{R1} - I_{R2} - I_{R3} = 0 \tag{2}$$
$$I_{R2} - I_{V2} = 0 \tag{3}$$

5. **Apply Ohm's law to each resistor ($I = V/R$).** Using Ohm's Law, write the current through each resistor in terms of the voltage across it. Write the voltage across each resistor in terms of the node potentials u_1, u_2, u_3, and ground. (Recall that ground has a potential of 0).

$$I_{V_1} - (u_1 - u_2)/R_1 = 0 \tag{4}$$
$$(u_1 - u_2)/R_1 - (u_2 - u_3)/R_2 - u_2/R_3 = 0 \tag{5}$$
$$-I_{V_2} + (u_2 - u_3)/R_2 = 0 \tag{6}$$

Note that when writing the voltage across a resistor in terms of node voltages, we write the higher potential node first. This is the *first* node the current passes through.

6. **Substitute known potentials for node voltages.** Since we know that between ground and node u_1 we have a voltage source V_1, we conclude that $u_1 = V_1$. Similarly, we can conclude that $u_3 = V_2$. This leaves us with only one unknown, u_2.

$$(u_1 - u_2)/R_1 - (u_2 - u_3)/R_2 - u_2/R_3 = 0 \tag{7}$$
$$(V_1 - u_2)/R_1 - (u_2 - V_2)/R_2 - u_2/R_3 = 0 \tag{8}$$

We can now solve for u_2:

$$u_2 = \frac{R_1 R_3 V_2 + R_2 R_3 V_1}{R_1 R_2 + R_1 R_3 + R_2 R_3} \tag{9}$$

Now, we have all the node potentials in the circuit!

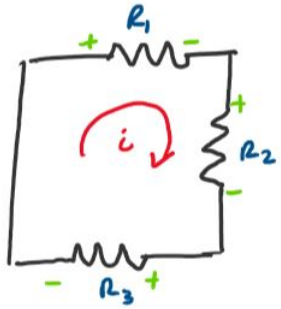
7. Now that we have all the node voltages, we may plug back into our Ohm's Law expressions to solve for all the currents. The circuit is thus fully solved!

2.5.3 Kirchoff's Voltage Law

Kirchoff's Voltage Law (KVL) provides us with useful information about voltage around loops in a circuit. Recall that a loop is any path in a circuit that starts and ends at the same point!
KVL tells us that the sum of voltage drops and rises around a loop must equal zero.
It may be useful to think back to our mountain analogy for an intuitive explanation of this law. Recall that in our analogy, voltage represented our elevation on the mountain.
What KVL says is that if we walk in a path on our mountain that takes us back to our starting point, our net change in elevation (voltage) is zero!
Just as the sum of elevation changes on our walk must be zero, the sum of voltage drops and rises around a loop must be zero.
Mathematically, KVL is expressed:

$$\sum_{loop} V_i = V_1 + V_2 + ... + V_n = 0$$

For each V_i a voltage change in a loop.
How can we derive KVL in a more mathematically rigorous sense? Outside of our intuition, where does it actually come from? Consider the following circuit, which is a basic loop:



A current, i runs through this circuit in the direction indicated (driven by some unknown force on the left-hand side). Note that the current through each

resistor is the same, since the resistors are in series.
We know that the total power dissipated is the sum of the powers dissipated in each element. We also know that energy for our system is conserved, since no outside power is being added to our circuit. Thus, the sum of all the powers must be 0.

$$P_1 + P_2 + P_3 = 0$$
$$iV_1 + iV_2 + iV_3 = 0$$

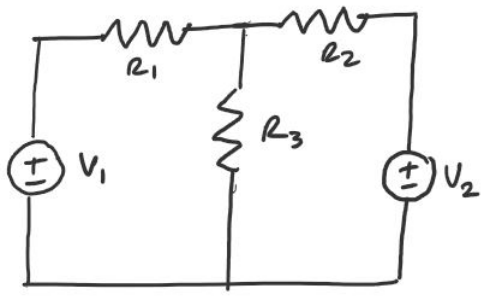
Now, let's factor out i and divide it out on both sides.

$$i(V_1 + V_2 + V_3) = 0$$
$$V_1 + V_2 + V_3 = 0$$

Thus, we find that the sum of voltages around the loop must be zero! This derivation may be generalized to any number of resistors and any type of electrical component.

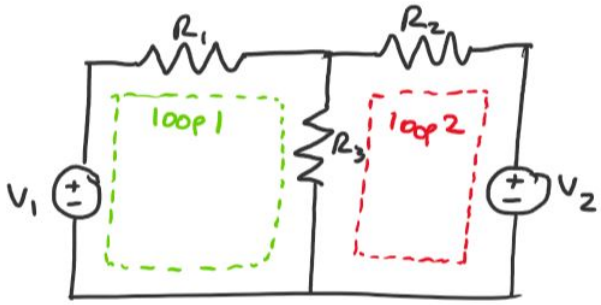
2.5.4 Applying KVL in Mesh Analysis

Just as we developed a problem-solving framework for KCL, we may also do so for KVL. In the following example, we solve the same problem as before, this time using a KVL procedure known as **Mesh Analysis**.
Ex: Solve for the currents and voltages across R1, R2, and R3

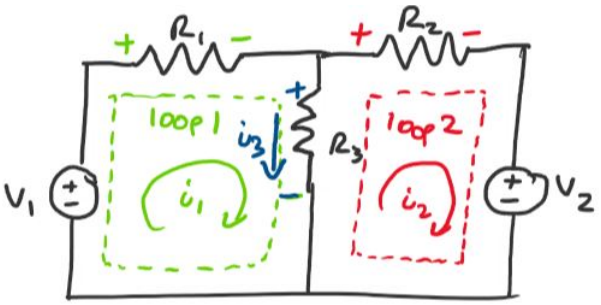


We now illustrate the procedure for analysis with KVL (Mesh Analysis):

1. **Identify the loops in the circuit.** Identify the paths in the circuit that begin and end at the same place. Note that there are multiple ways of finding loops - they'll all lead to equivalent equations.



2. **Assume a current direction in each loop.** Draw an arrow going clockwise (by convention) in each loop. This will be the current direction you assume inside the loop. Using this current direction, draw (+) and (-) signs on each resistor. Remember, current enters a resistor at (+) and exits at (-).



Make sure that if you have any resistors that are in two loops, that you define a *separate* current for that resistor (for example R_3). If we used i_1 and i_2 for R_3, we would end up with *conflicting* equations!

3. **Write the KVL equation for each loop.** Add all of the voltage changes in each loop. To keep track of the signs for each voltage, use the following convention: if the current arrow enters a component at the (+) terminal, add the voltage. If the current arrow enters a component at the (-) terminal, subtract the voltage. For this circuit, we would thus write:

$$-V_1 + V_{R_1} + V_{R_3} = 0 \tag{10}$$
$$V_{R_2} + V_2 - V_{R_3} = 0 \tag{11}$$

4. **Use Ohm's Law to rewrite KVL equations.** Using $V = IR$, rewrite the KVL equations for each loop. Note that for the resistor R_3, we define a separate current i_3, since we know the current through R_3 can't be both i_1 and i_2.

$$-V_1 + i_1 R_1 + i_3 R_3 = 0 \tag{12}$$
$$i_2 R_2 + V_2 - i_3 R_3 = 0 \tag{13}$$

5. **Use KCL to resolve "conflicting" components.** Since in this problem, we defined a separate current i_3 for R_3, we must now define i_3 in terms of the other currents to reduce the number of unknowns. To do this, we apply KCL at the node where the three resistors intersect to find:

$$i_3 = i_1 - i_2 \tag{14}$$

We may now rewrite our remaining KVL expressions by substituting:

$$-V_1 + i_1 R_1 + (i_1 - i_2) R_3 = 0 \tag{15}$$

$$i_2 R_2 + V_2 - (i_1 - i_2) R_3 = 0 \tag{16}$$

Now, we have two equations in two unknowns and may solve for the currents! The procedure is complete.

2.6 Measuring Circuit Quantities

Now that we have an understanding of what current, voltage, and resistance are - as well as how to relate them mathematically - we will briefly discuss methods of measuring them experimentally.

What's important to keep in mind is that when taking a measurement, we want to cause minimal disturbance to the original system while at the same time gain informative data. One challenge that we're faced with as engineers is that by measuring a system, we'll always be disturbing it from its natural state. Thus, there is no true perfect measurement!

Practically, current is measured with an **ammeter**. An ammeter is a measuring tool placed in series with the circuit element you wish to measure. To ensure that it has as little an impact on the circuit as possible, an ammeter has a very low input resistance.

Voltage is measured with a **voltmeter**. Voltmeters have wires that connect to either end of the component you wish to measure, and are said to be in parallel with the component. Because we want the current passing through the voltmeter's parallel branch to be as low as possible (to avoid disturbing the circuit we wish to measure), voltmeters have a very high input resistance.

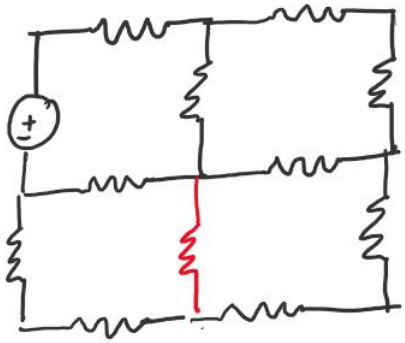
Resistance may be measured with an **ohmmeter**. Like voltmeters, ohmmeters are placed in parallel with the components. Similarly, they must have a high input resistance. To ensure the measurement will not be biased, all current and voltage sources must be disconnected before using an ohmmeter.

3 Superposition and Equivalence

Thus far, in studying Ohm's Law, Kirchoff's Laws, and the methods of Nodal and Mesh analysis, we've covered lots of ground in solving complex circuits. Although the methods we've developed so far have been very reliable, one common theme among them is that it can take a long time to reach a solution, especially for larger circuits. How does this impact us when we want to quickly test out different components?
One common problem in electronics involves testing a component of a different value and seeing how it impacts the rest of the circuit. What does this mean in terms of our circuit analysis? Would we have to re-solve the circuit every time we try out a different value?
To increase the efficiency of our circuit solving, we introduce the techniques of **equivalence** and **superposition**.

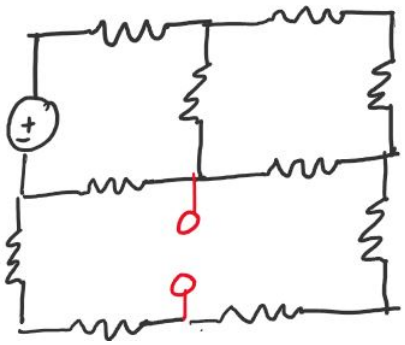
3.1 Equivalent Circuits

As mentioned in the introduction, it's a common problem to want to "try out" components of different values in a circuit. For example, in the diagram below, we might want to try out several resistances for the red resistor to see which resistance best suits our needs.



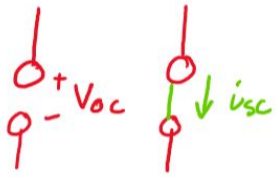
Above: A complex resistive network

When trying out different values for a component, we redraw the circuit to highlight our intentions. We remove the component remove from the circuit diagram and replace it with two **terminals**, which are simply the two nodes on either end of the component.
For the circuit above, we would redraw the diagram as follows:



Above: We redraw the circuit with two nodes in place of the component

Our goal is to develop a simplified **equivalent circuit** around the two nodes that's much easier to solve.
How do we ensure the simplified equivalent circuit has the same properties as our original circuit? We enforce two constraints.
First: the voltage across the two nodes should be the same in the equivalent and original circuits. Because there is nothing between the nodes, we call this the **open circuit voltage**.

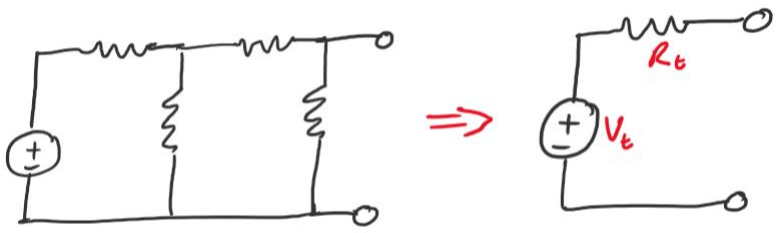


Open circuit voltage (V_{oc}) and short circuit current (i_{sc})

Second: the current across the nodes should be the same. When we connect a pure wire across the nodes, we want the current in our equivalent circuit to be the same as it would be in our original circuit. When we connect a wire between the two terminals, we refer to the current passing through the wire as the **short circuit current**.
Let's now introduce the two common equivalent circuits!

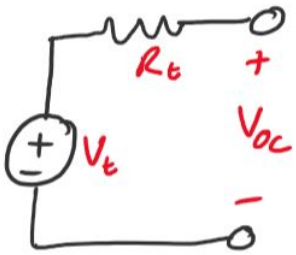
3.1.1 The Thevenin Equivalent

The Thevenin Equivalent Circuit replaces a complex circuit around two nodes with a circuit with a *voltage source and resistor in series.* A Thevenin Equivalent circuit appears as below:



Above: Transforming a complex circuit to a simpler Thevenin equivalent

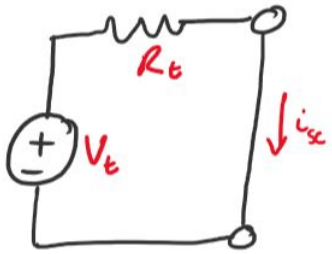
As you may see, we take a complex circuit with two open nodes and *transform* it into a much simpler Thevenin Equivalent with the same two open nodes.
Now that we've drawn out our simpler Thevenin equivalent circuit, we're faced with a problem: how do we find a voltage source and resistance that represent the *entire* original circuit?
Let's begin by solving for the **Thevenin voltage** (V_t), the voltage of the source in the Thevenin equivalent. We'll then proceed to find the **Thevenin resistance** (R_t), the resistance of the series resistor.
Let's first solve for the Thevenin voltage. We know from the first constraint we defined earlier that the open circuit voltage across the terminals in our Thevenin circuit *must* be the same as the open circuit voltage in our original circuit.



We also know that since the Thevenin equivalent circuit (above) is open, no current will flow through it. Applying KVL around our Thevenin circuit, we thus find:

$$-V_t + iR_t + V_{oc} = 0 \tag{17}$$
$$-V_t + 0R_t + V_{oc} = 0 \tag{18}$$
$$\boxed{V_t = V_{oc}} \tag{19}$$

Thus, the Thevenin voltage is equal to the open circuit voltage across the nodes in our original circuit! We may find this voltage from our original circuit by applying Nodal or Mesh Analysis just as we did with circuits in the past.
Let's now solve for Thevenin resistance. First, connect a wire across the terminals. Remember, when we connect a wire across the terminals, the current that flows is named the short circuit current.

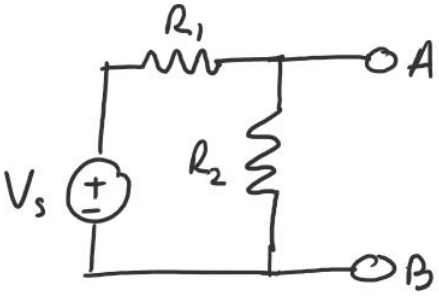


By KVL, we may now write:

$$i_{sc}R_t = V_t \tag{20}$$

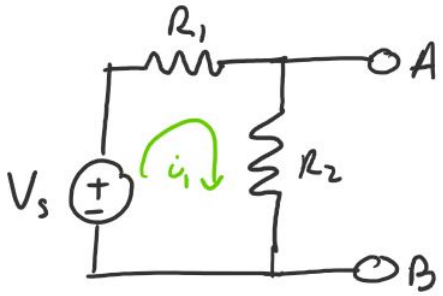
$$\boxed{R_t = V_t/i_{sc}} \tag{21}$$

Thus, the Thevenin resistance is equal to the Thevenin voltage divided by the short circuit current. Because of our second constraint, the Thevenin short circuit current is the same as the short circuit current in the original circuit. Thus, to find i_{sc}, we simply find the short circuit current for our original circuit.
We now have all the values we need to determine our Thevenin equivalent circuit! Let's illustrate the solution process with a short example.
Example: Find the Thevenin equivalent of the circuit below between the terminals A and B.



When solving for the Thevenin equivalent, we may use the following procedure:

1. **Solve for the open circuit voltage between the two terminals.** We may solve for V_{oc} by applying Kirchoff's laws to the circuit. In this case, V_{oc} is equal to the voltage across terminals A and B. Let's find this voltage by writing a KVL expression for the loop:



For the loop with current i_1, we write:

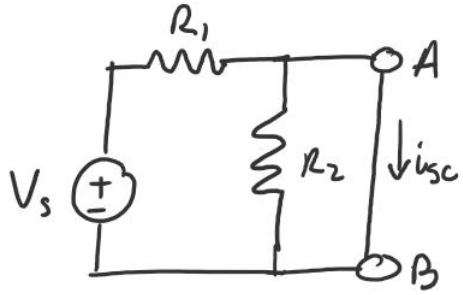
$$i_1 R_1 + i_1 R_2 - V_s = 0 \tag{22}$$
$$i_1(R_1 + R_2) = V_s \tag{23}$$
$$V_s/(R_1 + R_2) = i_1 \tag{24}$$

Noticing that the voltage between nodes A and B is the same as the voltage across resistor R_2, we use Ohm's Law and write:

$$V_{ab} = i_1 R_2 \tag{25}$$
$$V_{ab} = \frac{V_s R_2}{R_1 + R_2} \tag{26}$$

Since V_{ab} is the open circuit voltage across the terminals, we know $V_t = V_{ab}$. We now have the Thevenin voltage!

2. **Solve for the short-circuit current.** Now that we have the open-circuit voltage, the next step in finding our equivalent circuit is solving for the short-circuit current.
To get i_{sc}, connect a wire between terminals A and B and solve for the current that passes through it!



To solve for i_{sc}, we set up a KVL expression with a loop going around the entire outer circuit. Writing out the KVL equations, we get:

$$i_{sc} R_1 - V_s = 0 \tag{27}$$
$$i_{sc} = V_s/R_1 \tag{28}$$

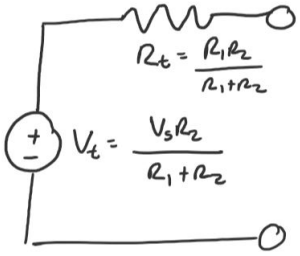
Now that we have the short-circuit current, there's only one step left: to plug back into Ohm's Law and solve for R_t.

3. **Use Ohm's Law to solve for** R_t. Now that we have the open circuit voltage and the short circuit current, we may use Ohm's Law to solve for the Thevenin resistance.

$$R_t = V_{AB}/i_{sc} = \frac{\frac{V_s R_2}{R_1+R_2}}{V_s/R_1} \tag{29}$$

$$R_t = \frac{R_1 R_2}{R_1 + R_2} \tag{30}$$

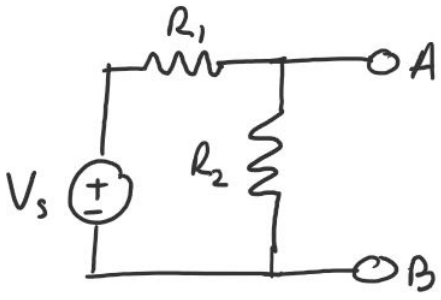
4. **Draw out the Thevenin Equivalent Circuit** Using V_t and R_t, draw out the equivalent circuit.



This final equivalent circuit has all of the same properties across the terminal as the original! Now, we may substitute in any resistor or component we like and solve for the voltage and current with ease.

3.1.2 The Voltage Divider Circuit

In the previous example, we found the Thevenin equivalent for one of the most famous and common circuit layouts: **the voltage divider**.



Above: The voltage divider circuit

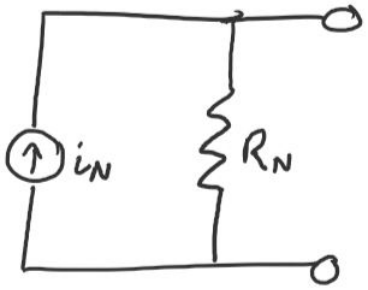
This circuit is gets its name from the following relationship, which we solved for in the previous example:

$$\boxed{V_{ab} = \frac{V_s R_2}{R_1 + R_2}} \tag{31}$$

As you can see, the voltage across terminals A and B is adjusted by *dividing* R_2 by $R_1 + R_2$.
This circuit will appear in many future examples - using the expression for V_{ab} above can greatly simplify your analysis. As such, look out for the voltage divider circuit as we move forward in electronics!

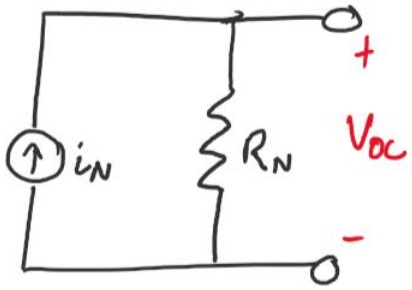
3.1.3 The Norton Equivalent Circuit

We now discuss the second type of equivalent circuit. Just as we defined an equivalent circuit with a voltage source and a resistor, we may also define an equivalent circuit with a *current source* and a resistor.
The **Norton Equivalent Circuit** is an equivalent circuit with a current source and a resistor in parallel. It appears in the form below:

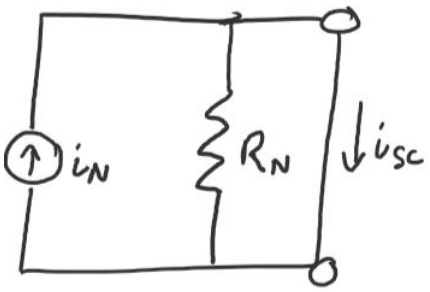


Above: A Norton Equivalent Circuit with Norton current i_N and Norton resistance R_N

Let's use Kirchoff's Laws to reach a conclusion on how to find i_N and R_N. First, let's take a look at the open-circuit voltage.



As we can see from the picture, the Norton open circuit voltage is simply the same as the open-circuit voltage for the larger circuit.
Now, let's look at the short-circuit current.



From a quick analysis of the short-circuit current, we may conclude:

$$\boxed{i_N = i_{sc}} \tag{32}$$

That is, the value of the current source in a Norton equivalent circuit is equal to the short-circuit current of the overall circuit.
To find the Norton resistance R_N, we use the same methodology that we did for Thevenin resistance: apply Ohm's Law. After a quick analysis, we come to the Ohm's Law expression:

$$\boxed{R_N = V_{oc}/i_{sc}} \tag{33}$$

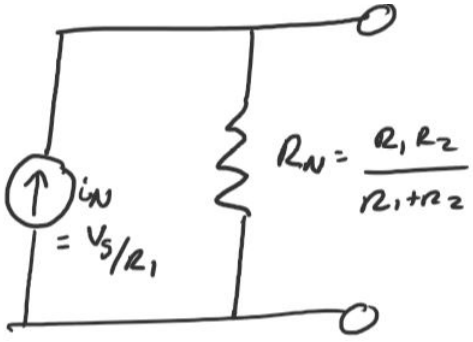
Since this is the same as the expression we found for Thevenin resistance, we conclude:

$$\boxed{R_N = R_t} \tag{34}$$

Thus, all of our Norton circuit variables are defined!
To find the Norton Equivalent for a circuit, we use a very similar process to the Thevenin Equivalent.

1. **Find the open circuit voltage.**
2. **Find the short-circuit current.** This will be equal to the Norton current.
3. **Apply Ohm's Law to find the Norton Resistance.**
4. **Draw the circuit diagram.**

The Norton Equivalent for the voltage divider circuit would thus be:



Notice the similarities in the procedures for finding the Thevenin and Norton equivalent circuits! Since we solve for all the same quantities for both, once we've found the Thevenin equivalent, we already have all the information we need to draw the Norton equivalent and vice-versa!

3.1.4 Finding R_t Directly

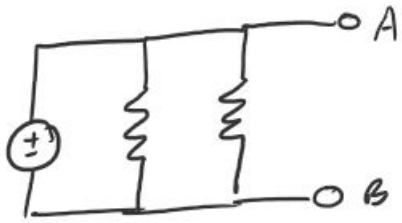
The method we've discussed thus far for finding the Thevenin and Norton properties - testing for open and short-circuit voltage and current - works very well for large networks of circuits.
However, for a small circuit, it might seem overly time consuming to solve both the open and short-circuited circuits. Is there an easier and faster way to solve for the properties of the Thevenin and Norton circuits?
The answer to this question lies in **equivalent resistance**. Recall from our past discussions that to find the equivalent resistance of n resistors in series, we add the resistances directly:

$$R_{series} = \sum_{i}^{n} R_i = R_1 + R_2 + ... + R_n \tag{35}$$

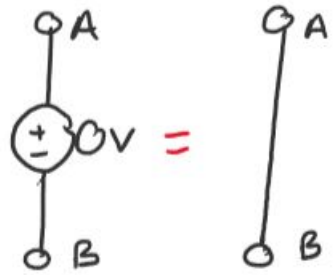
Also recall that to find the equivalent resistance of n parallel resistors, we sum up the reciprocals of the resistances and find the sum's reciprocal. That is:

$$1/R_{parallel} = \sum_{i}^{n} 1/R_i = 1/R_1 + 1/R_2 + ... + 1/R_n \tag{36}$$

To find the Thevenin resistance of a circuit, instead of solving for both V_{oc} and i_{sc} and dividing to get R, we can actually find the equivalent resistance of the circuit around the two terminals of interest.

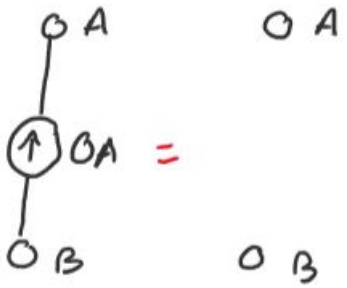


For example, the Thevenin resistance of the circuit above is equal to the equivalent resistance of the circuit outside of the two terminals. But, when we examine the circuit above, we encounter a challenge: how do we account for the voltage source when calculating equivalent resistance?
When finding the equivalent resistance of a circuit, we must **zero out** all sources. When we "zero out" a source, we set the value of the source to 0. How do we achieve this?
To zero out a voltage source, we replace the source with a **wire**. Because there is zero drop in voltage across a wire, this is the same as a voltage source of 0 volts.



Above: To set a voltage source to $0V$, we replace it with a wire

How do we zero out a current source? To set a current source to $0A$, we replace it with an open circuit. Since no current can flow across an open circuit, we know that this is equivalent to a $0A$ current source.

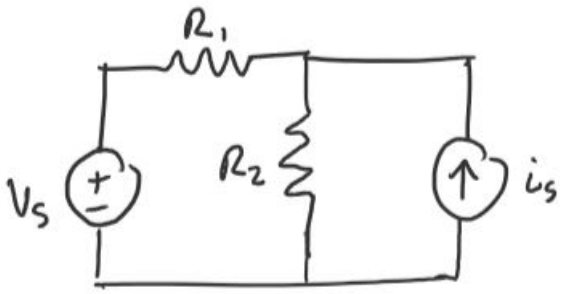


Above: To set a current source to $0A$, we replace it with an open circuit

After all the sources have been zeroed out, to find the Thevenin or Norton resistance, simply find the equivalent resistance of the remaining circuit.
How do we know which method to use to find the Thevenin or Norton resistance? When the circuit appears to be particularly complex, it's wise to use the more reliable $R = V_{oc}/i_{sc}$ method. This is due to the fact that it's oftentimes easier to make mistakes when finding equivalent resistance than it is with the more calculation-based KCL and KVL methods.
For smaller and simpler circuits, however, equivalent resistance can be a valuable tool to use!

3.2 Superposition

In our study of electronics thus far, we've investigated a variety of complex circuits. From large networks of resistors to multi-source circuits, we've developed an arsenal of tools which we may use to efficiently and accurately make conclusions.
Superposition is another valuable tool. We may use to simplify the analysis of circuits with multiple sources.



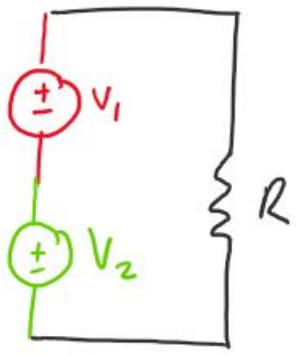
Above: An example of a circuit that may be solved via superposition

We've seen in the past that it can be challenging to solve such multi-source problems. There are lots of constraints to keep track of and it can become easy to neglect something in the problem-solving process.
The **principle of superposition** states that the total current and voltage across an element is equal to the sum of currents and voltages due to each source. In other words, if we solve the circuit with each source independently and sum the results, we'll get the total current and voltage for each element. Mathematically, for i_t and V_t the total current and voltage in each component, and for m the number of sources:

$$i_t = \sum_n^m i_n = i_1 + i_2 + ... + i_m \tag{37}$$

$$V_t = \sum_n^m V_n = V_1 + V_2 + ... + V_m \tag{38}$$

Let's illustrate this principle with a simple example. Observe the circuit below, which has two voltage sources.

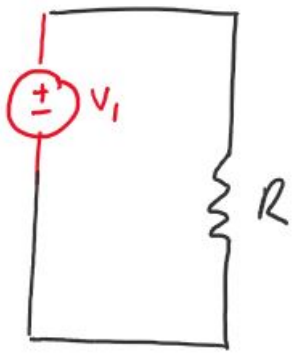


First, we'll solve this circuit using KCL and KVL for the *entire* circuit. Then, we'll treat each source separately, add the results, and see if we reach the same answer.

By KVL, we may find the current i_R and voltage V_R across the resistor to be:

$$i_R = (V_1 + V_2)/R \tag{39}$$
$$V_R = V_1 + V_2 \tag{40}$$

Now, let's solve the circuit source-by-source. Begin by zeroing out voltage source V_2 and concentrating on V_1. Recall that to zero out a voltage source, we replace it with a wire.



Now, we find the current and the voltage across the resistor using our circuit analysis techniques. For i_{R1} the current through the resistor due to V_1 and V_{R1} the voltage due to V_1:

$$i_{R1} = V_1/R \tag{41}$$
$$V_{R1} = V_1 \tag{42}$$

Doing the same analysis for voltage source V_2, we find:

$$i_{R2} = V_2/R \tag{43}$$
$$V_{R2} = V_2 \tag{44}$$

Now, let's add the results together and see what we get! For i_R the total current through the resistor and V_R the total voltage across the resistor:

$$i_R = (V_1 + V_2)/R \tag{45}$$
$$V_R = V_1 + V_2 \tag{46}$$

Thus, we see by summing the currents and voltages due to *each source*, we reach the same solution that we did by analyzing the circuit all at once! This is a basic example of solution by superposition.

3.2.1 Linearity and Superposition

Why does superposition work? To dive into this question, let's briefly discuss systems of equations.

You may remember from your linear algebra course that when you have a system of linear equations, you may add the equations together to find a new, equivalent system. For example, for the system below:

$$x_1 + x_2 = 3 \tag{47}$$
$$x_1 - 4x_2 = 7 \tag{48}$$

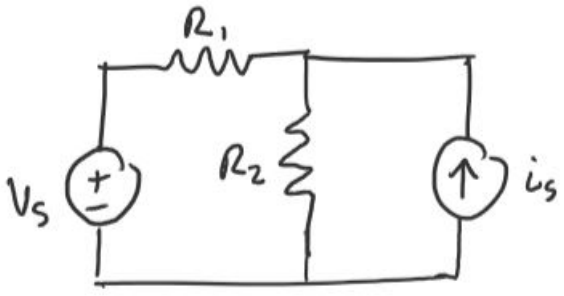
We can add the two equations together to produce a third equation with the same solution:

$$2x_1 - 3x_2 = 10 \tag{49}$$

The key property that makes this possible is **linearity**. Since the equations are *linear* (the highest exponent is 1), we can add and subtract equations without affecting the solution.
When we look back at all the resistive circuits we've analyzed, we find that all of the systems we've solved have been linear! This is because Ohm's Law, $V = IR$, is a linear equation.
This means that the system of equations we find from resistive circuits are just like any other linear system - we may add and subtract equations without affecting the solution.
The principle of superposition comes directly from this linearity. Because circuits are linear, we may find the equations from each source and simply *add* them to solve for the entire circuit.

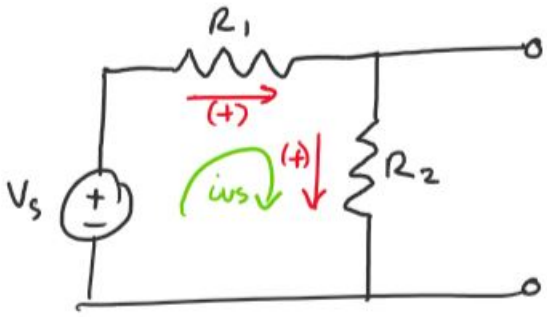
3.2.2 The Superposition Procedure

Now that we've defined the mechanics of superposition, let's develop a procedure for solving circuits with superposition. Let's follow the example below:
Solve for the currents in R_1 and R_2 using superposition:



Recognizing this circuit as a multi-source circuit (it has both a current and a voltage source), we may solve it via superposition.

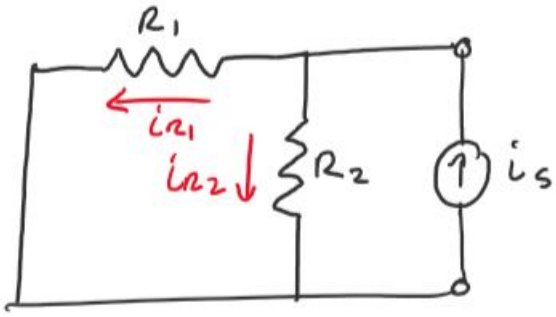
1. **Focus on the leftmost source. Zero out all other sources.** By convention, we begin by focusing on the leftmost source, which here is V_s. We then zero out all other sources in the circuit. To do this, we make the current source an open circuit so no current may flow through it.



2. **Solve the circuit with the first source.** In this example, by applying KVL around the first loop, we may find the current passing through R_1 and R_2. Label this i_{vs}, as it's for the first source, and make careful note of the direction. Label this direction as positive.

$$i_{vs}R_1 + i_{vs}R_2 - V_s = 0 \tag{50}$$
$$i_{vs}(R_1 + R_2) = V_s \tag{51}$$
$$V_s/(R_1 + R_2) = i_{vs} \tag{52}$$

3. **Focus on the next source in the circuit. Zero out all other sources.** Now, we need to zero out the voltage source and focus on the current source only. Recall that we zero out a voltage source by replacing it with a wire.



4. **Solve the new circuit.** Using KCL and KVL, we may find the currents through R_1 and R_2 due to the current source. First, let's write a KCL equation at the node where the resistors and current source meet. Then, let's apply KVL around the loop with the two resistors. Note that i_{R1} is in the *opposite* direction to what we defined as positive in step 2 - we'll account for this in the next step.

$$i_s - i_{R1} - i_{R2} = 0 \tag{53}$$
$$i_{R1}R_1 - i_{R2}R_2 = 0 \tag{54}$$

Solving, we get:

$$i_{R1} = i_{R2}R_2/R_1 \tag{55}$$
$$i_s = i_{R2}R_2/R_1 + i_{R2} \tag{56}$$
$$i_s = i_{R2}(R_2/R_1 + 1) \tag{57}$$
$$i_{R2} = \frac{i_s R_1}{R_2 + R_1} \tag{58}$$

Now, we substitute back in to get i_{R1}:

$$i_{R1} = i_s - i_{R1} \tag{59}$$
$$i_{R1} = i_s - \frac{i_s R_1}{R_2 + R_1} \tag{60}$$
$$i_{R1} = \frac{i_s R_2}{R_2 + R_1} \tag{61}$$

5. **Sum up the currents from each source to find total current.** To find the total current through each component, sum up the currents found for the component from each source, *making sure* to account for the directions of the current from each source! Note that since i_{R1} is opposite to what we defined in part 2, we subtract it from the part 2 current.

$$i_{R1t} = \frac{V_s}{R_1 + R_2} - \frac{i_s R_2}{R_2 + R_1} \tag{62}$$
$$i_{R2t} = \frac{V_s}{R_1 + R_2} + \frac{i_s R_1}{R_2 + R_1} \tag{63}$$

The circuit is now fully solved by superposition!

4 Capacitors, Inductors, and Nonlinearity

So far, we've completed a thorough analysis of linear resistive circuits. Through Ohm's law, as well as our circuit analysis techniques, we developed a strong foundation for solving circuits with resistors. But, what happens when resistors aren't the only components in a circuit? In this section, we'll learn how to analyze several other types of electronic components. Where we'll begin, however, is with a formal exploration of what it means for a circuit component to be linear. We'll then proceed to learn how we can deal with nonlinear components, and how their analysis differs.
Following this, we'll introduce the analysis of capacitors and inductors, and solve the differential equations they bring into circuit analysis.

4.1 Nonlinear Components

Thus far, when solving circuits, we've always been able to set up our circuit problems as a system of linear equations. As we discussed briefly in our review of superposition, this is because Ohm's Law, $V = IR$ is a linear equation.
At a more fundamental level, however, what does it mean to be linear? What are other, less obvious examples of linear relationships? How do we analyze circuits with *nonlinear* components?

4.1.1 Proving Linearity (Optional)

You may remember from your math classes that a **linear transformation** T is a transformation that satisfies the following relationship:
For a and b constants and x and y variables:

$$T(ax + by) = aT(x) + bT(y) \tag{64}$$

Let's apply this definition to prove that Ohm's Law is linear. First, define Ohm's Law as a transformation T that takes in current and returns voltage. T is then written:

$$T(I) = V = IR \tag{65}$$

To prove Ohm's Law linear with the above definition, imagine we have two currents I_1 and I_2 and constants a and b. Then:

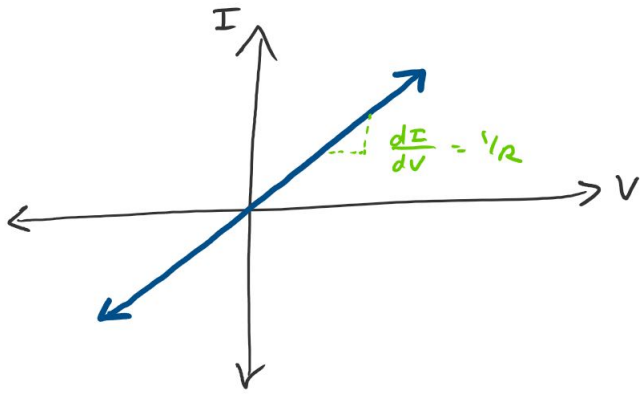
$$T(aI_1 + bI_2) = (aI_1 + bI_2)R \tag{66}$$
$$T(aI_1 + bI_2) = aI_1R + bI_2R \tag{67}$$
$$T(aI_1 + bI_2) = a(I_1R) + b(I_2R) \tag{68}$$
$$T(aI_1 + bI_2) = aT(I_1) + bT(I_2) \tag{69}$$

Thus, Ohm's Law satisfies linearity and is therefore a linear equation.
This proof is something we can do for all the relationships we'll encounter in ME 100. If we ever want to verify that a relation is linear, we may always use

the $T(ax + by)$ method.
We can also use this method to verify another important claim: *the derivative and the integral are both linear transformations.* This idea is something we'll refer back to over the course of this section.

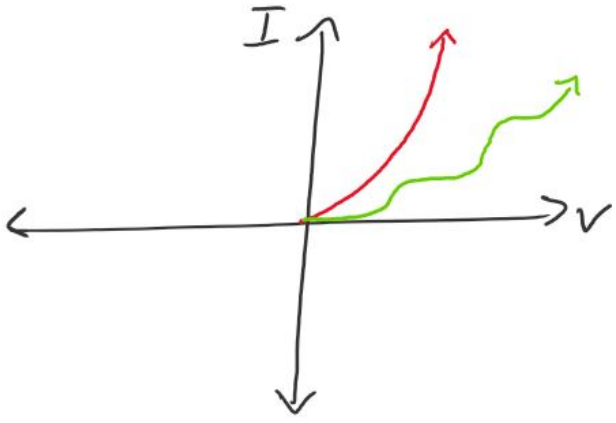
4.1.2 Load Line Analysis

In the past, all of our circuit components have been strictly linear. The resistor, for example, holds the following graphical relationship between voltage and current:



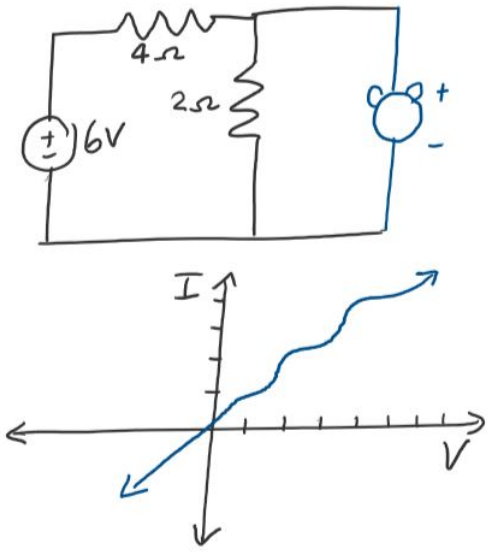
Above: A graph of Ohm's Law for a resistor

As mentioned before, this has allowed us to set up simple linear equations to represent our circuits, as well as employ techniques such as superposition to simplify our analysis.
What would happen, however, if the current-voltage relationship for a component was nonlinear and instead looked like one of the following?



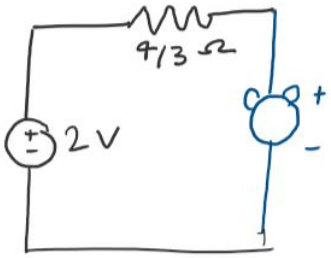
With such a component in the circuit, we would now be faced with a much more challenging, and in some cases - unsolvable - analysis.

Load line analysis is a graphical technique used to analyze circuits with nonlinear components. It can be used when we have a *single* nonlinear component in the circuit and a graph of the current-voltage relationship for that component.
We now illustrate the load-line analysis procedure with the following example: *The following circuit contains a nonlinear component, named the Oskinator. Find the current and voltage across the Oskinator using the current-voltage graph below.*

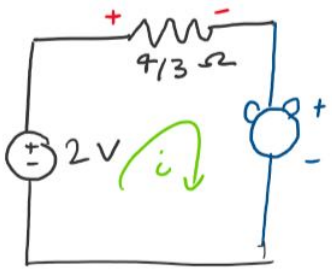


To find the current and voltage across a nonlinear component in a circuit, we use the following procedure:

1. **Find the Thevenin Equivalent around the nonlinear component.** When analyzing a circuit with a nonlinear component, we want to *isolate* the nonlinear component in the circuit. We do this by constructing the Thevenin Equivalent circuit around the nonlinear component.
   Note that the circuit drawn above is a *voltage divider* circuit. Solving for the Thevenin Equivalent, we find:

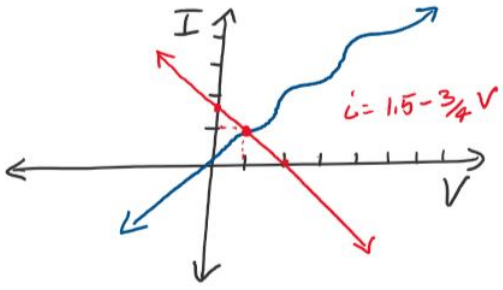


2. **Write a KVL equation using the Thevenin Equivalent**. Now that we have a simple circuit we can work with, we may write out a KVL equation to represent our circuit. Our goal is to solve the KVL equation for a *linear equation* for current as a function of voltage across the nonlinear component.



$$0 = -V_s + iR + V_{oski} \tag{70}$$
$$iR = V_s - V_{oski} \tag{71}$$
$$4/3i = 2 - V_{oski} \tag{72}$$
$$i = 1.5 - 3/4V_{oski} \tag{73}$$

3. **Plot the KVL equation on the IV graph and find the intersection.** Using the equation we solved for in step 2, which relates the current and voltage across the nonlinear component, we now plot the equation on the current-voltage graph for the nonlinear component provided for us in the problem statement. This graph of the KVL equation is called the **load line**, as it represents the load across the nonlinear component.
The intersection of our KVL equation with the provided graph will tell us the current and voltage across the nonlinear component!



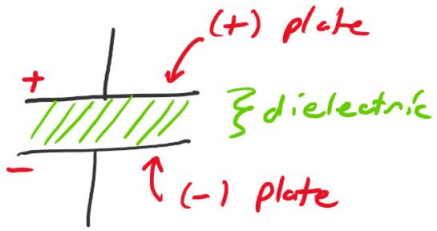
Thus, looking at the intersection of the load line with the current-voltage relationship of the Oskinator:

$$\boxed{i_{oski} = 1A} \tag{74}$$
$$\boxed{V_{oski} = 1V} \tag{75}$$

This procedure may be applied to *any* circuit with a single nonlinear component.

4.2 Capacitors

Thus far, we've talked about components that supply and dissipate electrical energy - are there components that *store* electrical energy as well?
Capacitors are a linear electrical component that store electrical energy between two conducting plates. When positive charges gather on the top plate, they induce a negative charge on the bottom plate, which generates an electric field. It's in this electric field that electrical energy is stored.
To increase the amount of charge a capacitor is able to store, the plates are often separated by a thin insulating material, known as a **dielectric**. This dielectric can be any insulating material from air to nylon!



As seen in the image above, the circuit symbol for a capacitor is two parallel lines with wires coming out of either plate.
How can we quantify how much charge a capacitor can store? The **capacitance** (C) of a capacitor measures how much charge per volt can be stored on the capacitor's plates. Capacitance is measured in units of Farads (F).
The relationship between charge, capacitance, and voltage across the plates is expressed by the following formula:

$$\boxed{Q = CV} \tag{76}$$

For Q the charge stored on each plate, C the capacitance, and V the voltage across the plates of the capacitor.

4.2.1 I-V Relationship for a Capacitor

As we've seen in our analysis of resistive circuits, Ohm's Law, which related the current and voltage across a resistor, was an incredibly useful equation.
Just as it was important to define Ohm's Law for a resistor, it's equally important to develop a current-voltage relationship for a capacitor.
Let's begin our derivation of the capacitor I-V relationship with our fundamental capacitor equation, $Q = CV$. Let's start by differentiating both sides of the

equation. Remember, capacitance (C) is a constant.

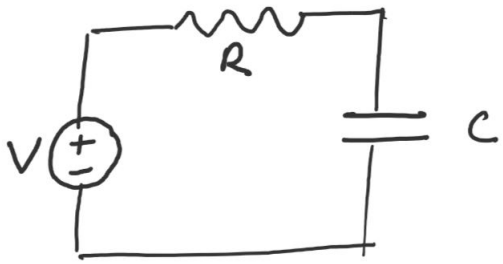
$$Q = CV \tag{77}$$

$$\frac{dQ}{dt} = C\frac{dV}{dt} \tag{78}$$

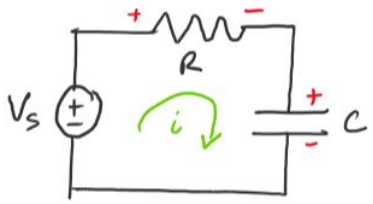
Now, we remember an important fact about the definition of current: $I = dQ/dt$. Substituting this relation into the above, we find:

$$\boxed{I = C\frac{dV}{dt}} \tag{79}$$

Let's examine some of the details of this equation. We find that if the voltage across a capacitor *doesn't* change ($dV/dt = 0$), then the current across the capacitor will be 0.
Before we proceed with analyzing circuits with capacitors, we must ask ourselves the important question: is the I-V relationship for a capacitor *linear*?
Since the capacitor equations have no exponents higher than 1 and the derivative is a *linear transformation*, we conclude that the I-V relationship for a capacitor is linear.
Now that we've established some of the basic properties of a capacitor, let's see how capacitors behave when placed in a circuit. The circuit below is an example of an **RC circuit**, a circuit with both a resistor and a capacitor.
Find the voltage across the capacitor as a function of time.



Note that now, since capacitors have *time* in their I-V relationship, RC circuits will not have constant currents and voltages, unlike purely resistive circuits.
We begin our analysis by applying KVL around the loop.



Using our KVL sign convention, we find:

$$V_R + V_c - V_s = 0 \tag{80}$$

Applying Ohm's law, we know that the voltage across the resistor is $V_R = iR$. We also know that since the resistor and the capacitor are in series, the current through them must be the same.
Thus, we may substitute in our new expression for the current across a capacitor: $i = C\frac{dV_c}{dt}$.

$$V_R = iR \tag{81}$$

$$i = C\frac{dV_c}{dt} \tag{82}$$

$$V_R = RC\frac{dV_c}{dt} \tag{83}$$

Now that we have the voltage across the resistor in terms of the voltage across the capacitor, we substitute into our KVL equation and divide by RC to get the resulting equation in a familiar form.

$$V_s = RC\frac{dV_c}{dt} + V_c \tag{84}$$

$$\frac{V_s}{RC} = \frac{dV_c}{dt} + \frac{V_c}{RC} \tag{85}$$

We now recognize this differential equation as one that may be solved by the *integrating factor* method. To review, if you have a differential equation of the form:

$$\frac{dy}{dx} + ay = b \tag{86}$$

For constants a, b, and c, the solution to this equation is of the form:

$$y(x) = \frac{b}{a} - ce^{-ax} \tag{87}$$

Keeping this in mind, we return to our capacitor differential equation and solve using the integrating factor method. Note that we use a constant of integration k to avoid confusion with capacitance C.

$$\frac{V_s}{RC} = \frac{dV_c}{dt} + \frac{V_c}{RC} \tag{88}$$

$$V_c(t) = \frac{V_s/RC}{1/RC} - ke^{\frac{-1}{RC}t} \tag{89}$$

$$V_c(t) = V_s - ke^{-t/RC} \tag{90}$$

We now have a general solution to the capacitor differential equation! To get the particular solutions to the differential equation, we'll consider two cases.

First, let's consider the boundary condition that up until t=0, there is no charge on the plates (the voltage source is disconnected), and that right at t = 0, the source is connected. This condition is known as *charging*.
Let's use this boundary condition to solve for k. By $Q = CV$, we know that the voltage at time zero is 0.

$$V_c(t) = V_s - ke^{-t/RC} \tag{91}$$
$$0 = V_s - ke^{0/RC} \tag{92}$$
$$0 = V_s - k \tag{93}$$
$$k = V_s \tag{94}$$

This tells us that for a capacitor being charged, $V_c(t)$ may be represented by the equation:

$$V_c(t) = V_{in} - V_s e^{-t/RC}$$
$$\boxed{V_c(t) = V_s(1 - e^{-t/RC})}$$

Let's examine this equation for the charging capacitor. First, let's find what the voltage across the capacitor will be at time $t = 0$. Plugging this into the equation, we find:

$$V_c(0) = 0 \tag{95}$$

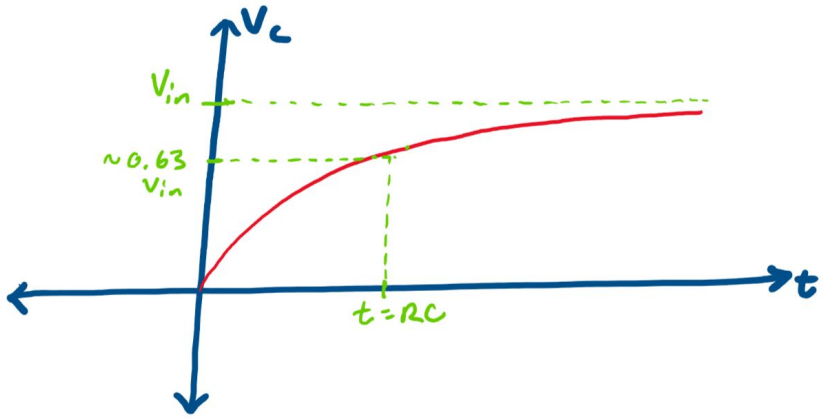
Since the capacitor is *uncharged* at $t = 0$, we may say that when a capacitor is uncharged, it acts like a *wire*. This is because the voltage drop across it is 0.
What about when $t \to \infty$? By taking the limit of $V_c(t)$ as $t \to \infty$, we find:

$$V_c(\infty) = V_s \tag{96}$$

This means that as time approaches infinity, and the capacitor becomes fully charged, the voltage across the capacitor equals the voltage of the source. But, using our KVL equation, this means that the voltage across the *resistor* in the circuit is 0. Thus, by Ohm's Law, at $t = \infty$, no current flows in the circuit.
Because of this, we conclude that a fully charged capacitor acts like an open circuit, because it doesn't allow any current to flow.
Let's now take a closer look at the time term in this equation, t/RC. At time $t = RC$, the fraction t/RC will equal 1. Because 1 is a simple value that's easy to work with, we give the time $t = RC$ a special name, the **time constant**, and a special symbol, τ.

$$\boxed{\tau = RC} \tag{97}$$

Plugging $t = \tau$ into the voltage equation, at $t = \tau$, $V_c \approx 0.63V_s$. Plotting $V_c(t)$ and τ on a graph, we find the charging curve for a capacitor looks like:

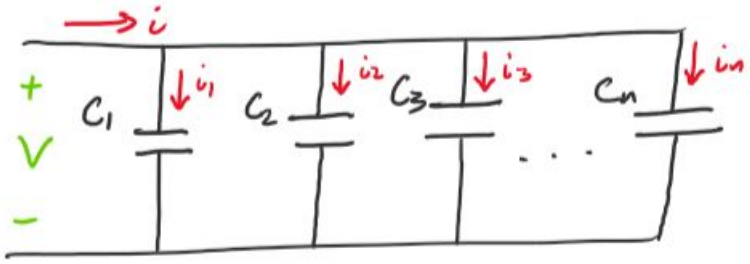


Above: $V_c(t)$ for a capacitor being charged. Notice how V_c is asymptotic to V_{in}.

The same process of solving for k may be done for the case when the voltage on the capacitor is at its maximum and the capacitor is fully charged. At this point, $V_c = V_{in}$. When the circuit is connected, the capacitor discharges, and V_c decreases in exponential decay.

4.2.2 Capacitors in Series and Parallel

Now that we've looked at how capacitors behave in simple circuits, it's natural to ask how we can find the **equivalent capacitance** of capacitors in series and parallel.
Recall that for series resistors, resistances added to get R_{eq}, while for resistors in parallel, 1/R added up to get $1/R_{eq}$. Let's examine how capacitance adds.
Let's consider the set of n capacitors in parallel:



To come to a useful conclusion about adding capacitance, we first apply KCL to the node just before the capacitors. We find from KCL:

$$i = \sum_{j}^{n} i_j = i_1 + i_2 + ... + i_n \tag{98}$$

Since we know for a capacitor, $i = C\frac{dV}{dt}$, and that the voltage across each capacitor is V, we make the following substitutions into our KCL expression:

$$i = \sum_j^n C_j \frac{dV}{dt} = C_1\frac{dV}{dt} + C_2\frac{dV}{dt} + ... + C_n\frac{dV}{dt} \tag{99}$$

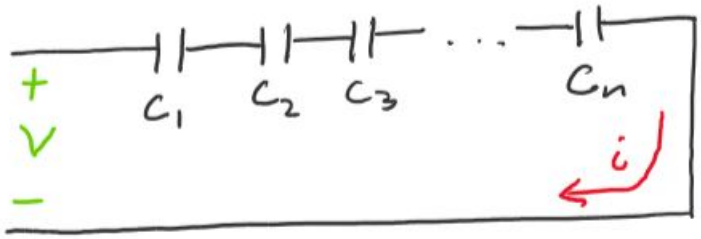
Now, factoring out dV/dt, we find:

$$i = (C_1 + C_2 + ... + C_n)\frac{dV}{dt} \tag{100}$$

Recognizing this as the capacitor I-V equation ($I = CdVdt$), we conclude that for n capacitors in parallel, the equivalent capacitance is:

$$C_{eq} = \sum_j^n C_j = C_1 + C_2 + ... + C_n \tag{101}$$

Thus, capacitors in *parallel* add like resistors in *series!*
Let's now perform a similar analysis for n capacitors in series.



We may begin by writing a KVL equation around the loop.

$$V = \sum_j^n V_j = V_1 + V_2 + ... + V_n \tag{102}$$

Now, we take the time derivative of both sides of this equation to get:

$$\frac{dV}{dt} = \frac{dV_1}{dt} + \frac{dV_2}{dt} + ... + \frac{dV_n}{dt} \tag{103}$$

Going back to our capacitor equation, $i = C\frac{dV}{dt}$, we know that $\frac{dV}{dt} = i/C$. Since each capacitor in the circuit is in series, the current through each is the same. Thus, we may write:

$$\frac{i}{C_{eq}} = \frac{i}{C_1} + \frac{i}{C_1} + \frac{i}{C_1} \tag{104}$$

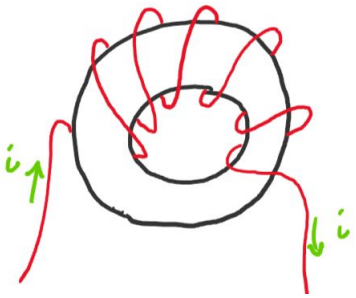
Factoring out i, we find that for n capacitors in series:

$$\frac{1}{C_{eq}} = \sum_j^n \frac{1}{C_j} = \frac{1}{C_1} + \frac{1}{C_2} + ... + \frac{1}{C_n} \tag{105}$$

This gives us the interesting result that capacitors in series add like resistors in parallel, and capacitors in parallel add like resistors in series.

4.3 Inductors

Another important linear circuit element is the **inductor**. Whereas a capacitor stores its electrical energy in an *electric field*, an inductor takes advantage of a *magnetic field*.
As we'll soon see, there are many similarities in the methods we use to solve circuits with inductors and capacitors.
Physically, inductors are coils of wire that are tightly wrapped around a core, which is typically made of a magnetic material such as iron. The current traveling through the wires *induces* a magnetic field in the core, which then comes back and produces an effect on the current.



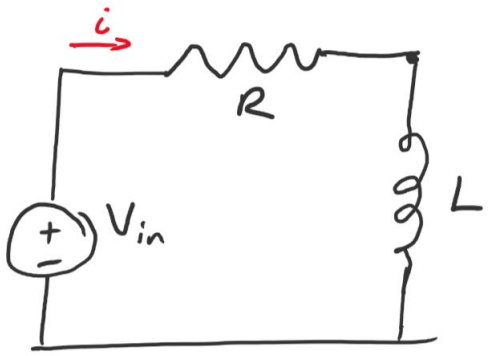
Above: an example of an inductor - a coil wrapped around a core

4.3.1 I-V Relationship for an Inductor

Using the laws of electromagnetism, the following relationship may be derived for an inductor:

$$\boxed{V = L\frac{dI}{dt}}$$

Notice the interesting similarity of this equation to the current-voltage relationship for a capacitor! Instead of capacitance, inductors have **inductance** (L), which is measured using the Henry (H).
Once again, since the derivative is a *linear transformation*, the current-voltage relationship for an inductor is *linear*.
Let's now use this inductor differential equation to examine how an inductor functions in the simple RL (Resistor-Inductor) circuit below. Note the coil-like symbol for an inductor.



We now follow a procedure that's much the same as that for the capacitor. First, since we know an expression for the voltage across an inductor ($V = LdI/dt$), we begin by writing a KVL expression for the circuit.

$$V_s = V_R + V_L \tag{106}$$

Substituting in Ohm's Law for the resistor and the inductor expression for the inductor, we find:

$$V_s = iR + L\frac{di}{dt} \tag{107}$$

$$\frac{V_s}{L} = i\frac{R}{L} + \frac{di}{dt} \tag{108}$$

We now recognize that this differential equation is of the *same* form as the capacitor equation. It may thus also be solved using the integrating factor formula. Applying this formula, we find (for some constant of integration k):

$$i(t) = \frac{V_s}{R} - ke^{-\frac{R}{L}t} \tag{109}$$

Let's solve for k for the case that at t=0, the voltage source is disconnected and current is 0.

$$0 = \frac{V_s}{R} - ke^0 \tag{110}$$

$$k = \frac{V_s}{R} \tag{111}$$

Substituting back into our $i(t)$ equation and factoring out V_s/R, we arrive at the formula:

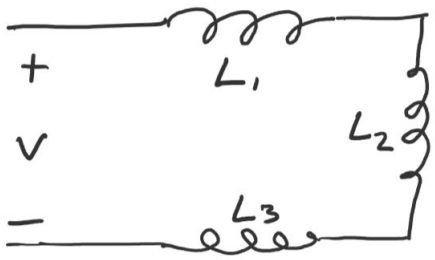
$$i(t) = \frac{V_s}{R}(1 - e^{-\frac{R}{L}t}) \tag{112}$$

Notice how similar this formula for $i(t)$ is to the $V_c(t)$ formula for a charging capacitor!

Due to this similarity, just as we defined the capacitor time constant, we may also define the **inductor time constant**. Recall: the time constant is the value of time that makes the exponent equal to 1.
Noting that the exponential term here is $\frac{R}{L}t$, we conclude by the same methodology as for the capacitor that the inductor time constant is L/R.

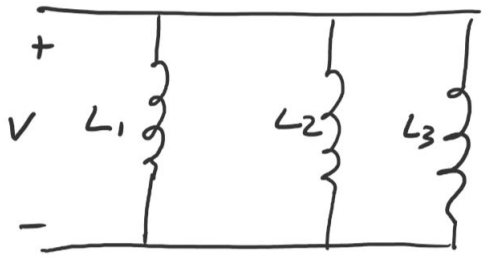
4.3.2 Inductors in Series and Parallel

As with capacitors and resistors, it can be helpful to determine the **equivalent inductance** of inductors in series and parallel. For the sake of brevity, the derivations have been omitted in this section.
Using a similar derivation to series capacitance, we find for inductors in series, inductance adds:



$$L_{eq} = \sum_{j}^{n} L_j = L_1 + L_2 + ... + L_n$$

For inductors in parallel, 1/inductance adds:



$$\frac{1}{L_{eq}} = \sum_{j}^{n} \frac{1}{L_j} = \frac{1}{L_1} + \frac{1}{L_2} + ... + \frac{1}{L_n}$$

Note that equivalent inductance is calculated in the same way as equivalent resistance!

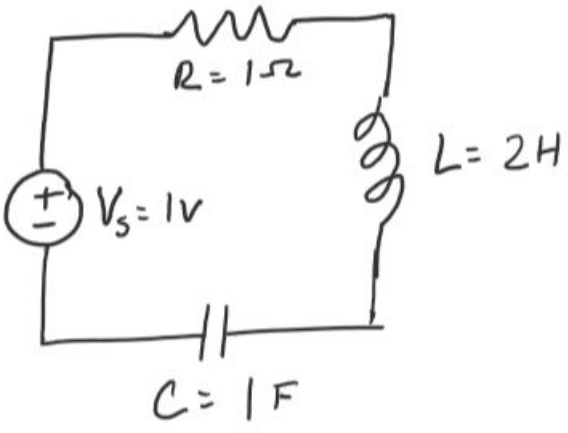
4.4 RLC Circuits

In introducing inductors and capacitors into our circuits, we've seen the increased complexity that comes with circuits whose quantities change in time. Instead of voltage and current being simple constant values, we've found them to change with time.

However, thus far, we've only considered circuits with capacitors and inductors in isolation. What happens if we combine resistors, inductors, and capacitors in a single **RLC circuit**?

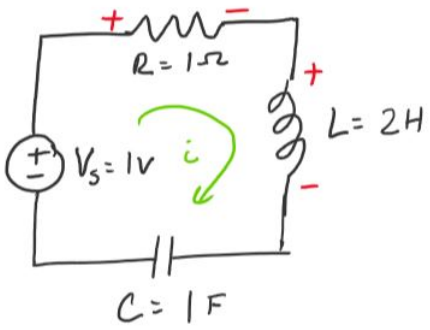
To answer this question, we now present an analysis of a series RLC circuit. As we develop the equations for an RLC circuit, take note of the intricate structure that can be achieved through simple electronic components!

Draw a graph of the current in the circuit versus time. Assume that before time $t = 0$, the capacitor is uncharged and zero current flows through the circuit. At time $t = 0$, the voltage source is connected.



Above: the series RLC circuit.

We may begin by writing a KVL expression around the loop. Assuming the current direction in the drawing below, we write:



$$V_s = V_R + V_L + V_c \tag{113}$$

Now, substituting the voltage-current relationship into each term, we find:

$$Ri + L\frac{di}{dt} + \frac{1}{C}\int_0^t i(\tau)d\tau = V_s \tag{114}$$

Note that τ is used as a "dummy variable" for t in the integral, and that the integral is derived from the capacitor expression $i_c = C\frac{dV_c}{dt}$.
To solve this challenging equation, we must *change variables* and turn it into a familiar form.
We begin this process with the definition $I = dQ/dt$. Using this definition, we may change this differential equation from an equation in terms of current to an equation in terms of charge! We now write:

$$R\frac{dQ}{dt} + L\frac{d^2Q}{dt^2} + \frac{1}{C}Q = V_s \tag{115}$$

Rearranging and dividing by L, we now recognize this equation as a *linear second order ODE*.

$$\frac{d^2Q}{dt^2} + \frac{R}{L}\frac{dQ}{dt} + \frac{1}{LC}Q = \frac{V_s}{L} \tag{116}$$

Now, we follow the solution procedure for a second order ODE. First, we find the characteristic polynomial and substitute in our given values:

$$s^2 + \frac{R}{L}s + \frac{1}{LC} = 0 \tag{117}$$

$$s^2 + \frac{1}{2}s + \frac{1}{2} = 0 \tag{118}$$

Now, using the quadratic formula, we may solve for the roots of the characteristic polynomial, which are complex (note that we use $\sqrt{-1} = j$):

$$s = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{119}$$

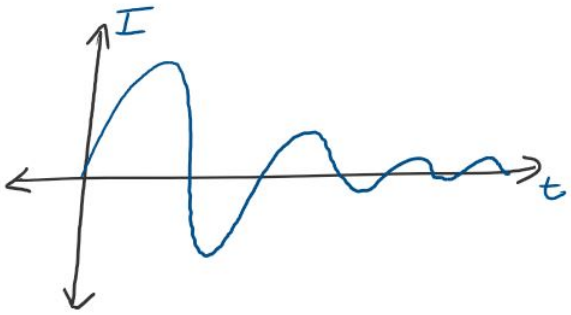
$$s = -\frac{1}{4} \pm j\frac{\sqrt{7}}{4} \tag{120}$$

Using ODE theory and this result, we can infer several things about the circuit:

1. Because the roots are complex, there will be sines and cosines in the solution.

2. Because the real parts of the roots are negative, there will be an exponential decay in current as time goes on.

With these two constraints, we may infer that the current in the circuit may be represented by a graph similar to:



Using the solution for a second order linear differential equation, we know from the roots of the characteristic equation that:

$$q(t) = e^{-1/4t}(c_1 \sin(\tfrac{\sqrt{7}}{4}t) + c_2 \cos(\tfrac{\sqrt{7}}{4}t) \tag{121}$$

This is the general solution to the RLC circuit problem. By differentiating our $q(t)$ expression, we may convert back to current. Following this, we may apply our boundary condition that at $t = 0$, voltage $V = V_s$.
Following through with this analysis, we find that for an RLC circuit, the solution will be of the form:

$$i(t) = C_1 e^{-\alpha t} \cos(\omega t) + C_2 e^{-\alpha t} \sin(\omega t) \tag{122}$$

For α, ω, C_1, and C_2 constants.

4.5 Parasitic Resistance, Capacitance, and Inductance

Although in electronics, we often think of resistors, capacitors, and inductors as separate components, in the real world, this isn't always the case.
Although a resistor is built to have only a resistance, due to material properties and imperfections, there is always some small capacitance and inductance in every resistor. These unintentional capacitances and inductances are said to be **parasitic**.
Similarly, a capacitor may have a parasitic resistance and inductance, and an inductor a parasitic resistance and capacitance.
Since in our models of circuits, we deal primarily with *ideal* components, this is something we consider to be negligible. However, in the real world, although the effects are typically small, parasitic resistance, capacitance, and inductance can be something important to consider.

5 Introduction to Phasors

So far, we've developed a number of powerful techniques for circuit analysis. We've looked at KCL and KVL, and have defined laws for how current and voltage interact in a number of circuit components.
In our analyses so far, we've always relied on one fact: that the signal supplied by our source is constant. In this section, we'll generalize what we've learned about constant (DC) signals to the world of sinusoidal (AC) signals.
Along the way, we're going to make use of complex numbers to make exciting conclusions about how sinusoidal signals can be represented and manipulated.

5.1 Complex Numbers

Before we jump into the details of AC circuit analysis, it's important to have a strong understanding of complex numbers. Let's briefly review some of the fundamentals.
First: what *is* a complex number? As you may remember from your math courses, a complex number is a number that may be expressed in the form:

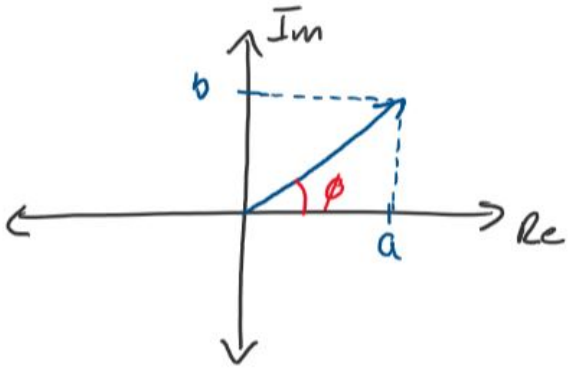
$$n = a + bj \tag{123}$$

For a and b real-numbered constants and $j = \sqrt{-1}$. Something interesting to note here is that unlike in mathematics, where we called $\sqrt{-1}$ "i," in electronics, we use j to avoid confusion with current.
Based on this definition, $n = a + bj$, every complex number has a *real* and *imaginary* part, defined as follows:

$$Re(n) = a \tag{124}$$

$$Im(n) = b \tag{125}$$

How may we represent a complex number outside of this simple notation? As it happens, there are lots of useful and insightful ways to think about complex numbers. One of these ways is to think of a complex number as a **vector** that extends from the origin. Using the x-axis as the real component and the y-axis as the complex component, we may graph a complex number as follows:



As you can see, the complex number is represented by a vector of length $\sqrt{a^2+b^2}$ positioned at an angle phi (ϕ) to the real axis. Using this graph, we can derive the following useful relationships:

$$|n| = \sqrt{a^2+b^2} \tag{126}$$

$$\phi = \tan^{-1}\left(\tfrac{b}{a}\right) \tag{127}$$

$|n|$ is known as the **magnitude** of the complex number, while ϕ is known as the **phase**.
Now that we've defined some key quantities for a complex number, let's discuss an important relationship known as **Euler's Identity**. Euler's identity for a complex number expresses the following surprising fact:

$$\boxed{e^{j\theta} = \cos(\theta) + j\sin(\theta)} \tag{128}$$

What this identity tells us is that complex numbers are intricately linked to sinusoidal, oscillating functions through the exponential function e^x.
Where does this formula come from? Let's write a short proof for Euler's Identity to find out. We begin our proof of this statement by looking at the Taylor expansions of the exponential, sine, and cosine functions:

$$e^{\theta} = 1 + \theta + \frac{\theta^2}{2!} + ... + \frac{\theta^n}{n!} \tag{129}$$

$$\sin\theta = \theta - \frac{\theta^3}{3!} + \frac{\theta^5}{5!} - ... \tag{130}$$

$$\cos\theta = 1 - \frac{\theta^2}{2!} + \frac{\theta^4}{4!} - ... \tag{131}$$

Let's take the Taylor expansion of $e^{j\theta}$ and see if we find any interesting information.

$$e^{j\theta} = 1 + j\theta - \frac{\theta^2}{2!} - j\frac{\theta^3}{3!} + \frac{\theta^4}{4!} + ... \tag{132}$$

Separating the real and complex terms of this expansion, we write:

$$e^{j\theta} = (1 - \frac{\theta^2}{2!} + \frac{\theta^4}{4!} - ...) + j(\theta - \frac{\theta^3}{3!} + ...) \tag{133}$$

We recognize the real part of this expansion as the Taylor expansion of $\cos\theta$ and the imaginary part as the Taylor expansion of $\sin\theta$. Thus, substituting in the sine and cosine functions for their expansions, we conclude:

$$e^{j\theta} = \cos\theta + j\sin\theta$$

And the identity is thus proved!
Because $e^{j\theta} = \cos\theta + j\sin\theta$ matches the form $n = a + jb$, Euler's Identity allows

us to write complex numbers in **exponential form**. The exponential form of a complex number is:

$$n = |n|e^{j\phi} \tag{134}$$

For $|n|$ the magnitude of the complex number n and ϕ the phase.
Now that we have several methods for representing complex numbers, let's discuss a valuable "counterpart" that each complex number holds. Every complex number $n = a+bj$ has a **complex conjugate** $\overline{n} = a-bj$. The complex number and its conjugate have the useful property that $n \cdot \overline{n} = a^2 + b^2$. Let's solve for this property:

$$n = a + bj \tag{135}$$
$$\overline{n} = a - bj \tag{136}$$
$$n \cdot \overline{n} = (a + bj)(a - bj) \tag{137}$$
$$n \cdot \overline{n} = a^2 + abj - abj - b^2 j^2 \tag{138}$$
$$n \cdot \overline{n} = a^2 + b^2 \tag{139}$$

When might we want to make use of this property? When simplifying fractions of complex numbers, we often use the complex conjugate to *remove* complex numbers from the denominator. For example, if we want to simplify the following:

$$n = \frac{1 + 2j}{2 - 4j} \tag{140}$$

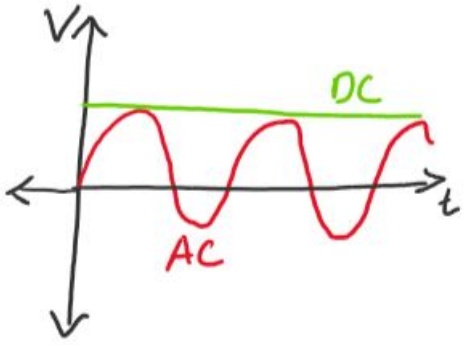
We may multiply the numerator and the denominator of the fraction by the complex conjugate of the denominator, $2 + 4j$.

$$n = \frac{1 + 2j}{2 - 4j} \cdot \frac{2 + 4j}{2 + 4j} \tag{141}$$
$$n = \frac{(1 + 2j)(2 + 4j)}{2^2 + 4^2} \tag{142}$$
$$n = \frac{-6 + 8j}{20} \tag{143}$$

As you can see, we've eliminated complex numbers from the denominator and have simplified n into an easier form.

5.2 Phasors

Now that we've developed the necessary background for complex numbers, we can begin to explore AC circuit analysis. AC analysis is centered around the question: what happens if our current and voltage sources *aren't* constant?
Instead of outputting a constant voltage such as $V = 10V$ or a constant current such as $I = 2A$, an AC source puts out an oscillating current or voltage, such as in the graph below:

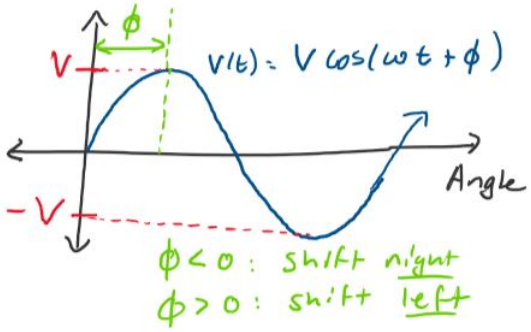


Above: an AC voltage signal (in red)

In general, we express all **sinusoidal** AC currents and voltages in the form:

$$\boxed{V(t) = V_s \cos(\omega t + \phi)} \tag{144}$$

For V_s the **amplitude** of the wave in units of Volts or Amps, ω the **angular frequency** in radians/second, and ϕ the **phase** of the wave in radians or degrees.



Note that if you're given a voltage in terms of the sine function instead of cosine, you should convert it to the cosine standard form with the following equation:

$$\sin(\theta) = \cos(\theta - 90^\circ) \tag{145}$$

Let's quickly think back to our analysis of circuits with capacitors and inductors. Recall that when capacitors and inductors are introduced to a circuit, we encounter challenging differential equations, such as the one below:

$$Ri + L\frac{di}{dt} + \frac{1}{C}\int_0^t i(\tau)d\tau = V_s \tag{146}$$

With sinusoidal voltage and current sources, these equations would only become more intimidating to solve!

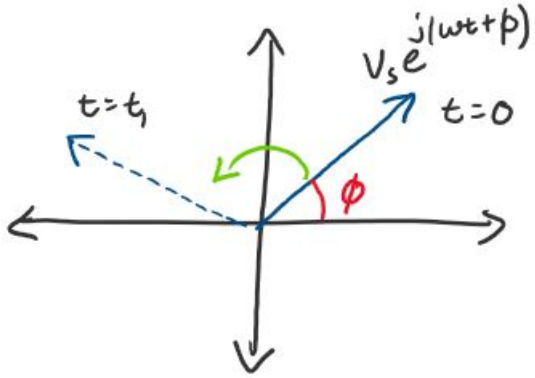
Thus, when dealing with AC sources, we'd like to find a way to *avoid* differential equations entirely. The answer to how we can do this lies in **phasor analysis**, which relies on complex numbers to avoid the need for differential equations.
Let's now develop the tools we need for phasor analysis. First, imagine we have some sinusoidal signal as a function of time in standard AC form:

$$V(t) = V_s \cos(\omega t + \phi) \tag{147}$$

Looking back to Euler's Identity, $e^{j\theta} = \cos\theta + j\sin\theta$, we notice something interesting: the standard form for voltage is equal to the *real* part of Euler's identity.

$$V(t) = V_s \cos(\omega t + \phi) = Re(V_s e^{j(\omega t+\phi)}) \tag{148}$$

Using Euler's Identity, we've now jumped from a sinusoidal function to an exponential function. Let's take a closer look at the exponential function and see if we can come up with a graphical interpretation.
Thinking back to the idea that a complex number $a + bj$ may be graphed as a vector, and that $a + bj = e^{j\theta}$, we now realize that the formula $V_s e^{j(\omega t+\phi)}$ represents a vector that *rotates* around the origin with time.



This vector rotates with an angular velocity of ω radians/sec, and at time $t = 0$, has a phase angle of ϕ.
Let's continue developing our mathematical representation by expanding the term in the exponent.

$$V(t) = Re(V_s e^{j(\omega t+\phi)}) \tag{149}$$
$$= Re(V_s e^{j\omega t+j\phi}) \tag{150}$$

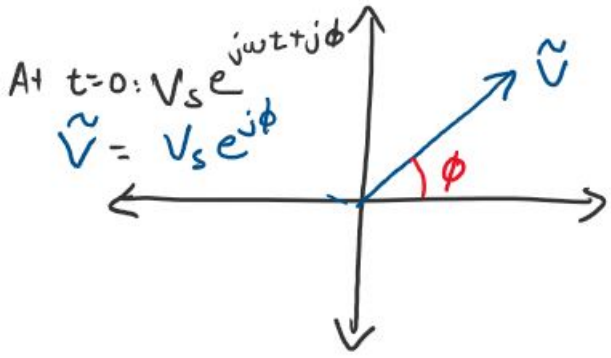
Now, by using the property of the exponential function: $e^{a+b} = e^a e^b$, we find:

$$V(t) = Re(V_s e^{j\phi} \cdot e^{j\omega t}) \tag{151}$$

Before we proceed further, let's discuss an interesting property of AC circuits. In any AC circuit with linear components (resistors, capacitors, and inductors), the frequency ω will *never* change. We can verify this by applying the current-voltage relationship for each resistor an observing that the frequency is unaffected by each component.
What does this mean for the equation we just defined? If frequency ω never changes, then we can solve the circuit just by focusing on the *phase* and the *magnitude*. As such, we take the equation $V(t) = Re(V_s e^{j\phi} \cdot e^{j\omega t})$ and focus on the term:

$$\boxed{\tilde{V_s} = V_s e^{j\phi}} \tag{152}$$

This term is known as a **phasor** (written as $\tilde{V_s}$). A phasor extracts all of the information we need for circuit analysis from a signal.
In particular, the form $V_s e^{j\phi}$ is known as the **exponential form** of the phasor. We may use it to describe the characteristics of a voltage source that oscillates with time. V_s is the *magnitude* of the voltage source, and ϕ is the phase.
Note that phasors are always written with a "tilde" $\sim$ or an "overline" $\overline{V}$ above the letter.
Thinking back to the graph we plotted earlier, we realize that the phasor for voltage is simply the complex voltage vector at time $t = 0$.



Now that we have a streamlined description of our signal, we can use the rules of complex numbers to rewrite our phasor in other forms. The first form we'll define is the **standard form**. When in standard form, the phasor $V_s e^{j\phi}$ is written:

$$\boxed{\tilde{V_s} = V_s \angle \phi} \tag{153}$$

Using the complex number identities we defined earlier, we may also convert the phasor to **complex form**.

$$\boxed{\tilde{V}_s = a + bj} \tag{154}$$

$$a = V_s \cos\phi \tag{155}$$

$$b = V_s \sin\phi \tag{156}$$

If we have a phasor in complex form and we'd like to convert *back* to standard form, we may use the complex number identities:

$$V_s = \sqrt{a^2 + b^2} \tag{157}$$

$$\phi = \tan^{-1}(b/a) \tag{158}$$

Let's now work through an example to convert from a sinusoidal function to its phasor.
Find the standard form phasor of the voltage $V(t) = 10\cos(100t + 45^\circ)$.
When converting from $V(t)$ to phasor form, we use the following procedure:

1. **Check that $V(t)$ is in standard form.** Recall that signals must be expressed in the standard form $V(t) = V_s\cos(\omega t + \phi)$ to use to the phasor representation. If a sine function is provided instead of the cosine function, convert to cosine by subtracting 90° from the phase. In this case, the provided equation is in the correct form.

2. **Identify the magnitude and phase.** The magnitude, $10V$, is the coefficient of the cosine function. The phase, 45°, is the angle added to the time term in the cosine function.

3. **Substitute the terms into the standard form equation.** Using the magnitude and phase found in step 2, substitute into the standard phasor form equation, $\tilde{V}_s = V_s\angle\phi$. This gives us our answer:

$$\tilde{V}_s = 10\angle 45^\circ \tag{159}$$

When using phasors to solve circuits, *all* of the circuit rules we've discussed so far (Ohm's law, KCL, KVL, superposition, equivalence) still apply! But, now that we've removed time from our equations, how do we define the impact that capacitors and inductors have on our circuit?

5.3 Impedance

To define the effects of resistors, capacitors, and inductors on AC circuits, we define the **impedance** of a component. Impedance may be thought of as resistance but for *all* linear circuit components. We define impedance (Z) as:

$$\boxed{Z = \frac{\tilde{V}}{\tilde{I}}} \tag{160}$$

In words, impedance is the phasor for voltage divided by the phasor for current. Note the similarity of this expression to Ohm's Law, which may be expressed as $R = V/I$. Instead of just resistors, however, impedance applies to capacitors and inductors as well.
Let's now solve for the impedance of our linear circuit components. First, we evaluate impedance for a resistor, which is trivially found through Ohm's Law.

$$Z_R = \frac{\tilde{V}}{\tilde{I}} \tag{161}$$

We recognize through Ohm's Law that $\tilde{V}/\tilde{I}$ is equal to R. Thus, a resistor's impedance is the *same* as its resistance.

$$\boxed{Z_R = R} \tag{162}$$

Let's now carry out a similar analysis to find the impedance of a capacitor. This time, let's begin with the equation for current across a capacitor:

$$\tilde{I} = C\frac{d\tilde{V}_c}{dt} \tag{163}$$

Now, we're faced with an interesting challenge: we have the take the derivative of a phasor! To do this, we start by converting V_c into complex form. Recall that a sinusoidal voltage may be represented by the equation:

$$V_c(t) = Re(V_s e^{j(\omega t+\phi)}) \tag{164}$$

$$V_c(t) = Re(V_s e^{j\omega t} e^{j\phi}) \tag{165}$$

Taking the time derivative of this equation, we find:

$$\frac{dV_c}{dt} = Re(j\omega V_s e^{j\phi} e^{jwt}) \tag{166}$$

Now, we find the *phasor* of this term. Recall that to find the phasor, we simply find the *coefficient* of the e^{jwt} term. This leaves us with the equation:

$$\frac{d\tilde{V}_c}{dt} = j\omega V_s e^{j\phi} \tag{167}$$

We've now successfully found the derivative of our phasor for voltage. Now, all that's left is to substitute back into our current-voltage relationship. Substituting, we find:

$$\tilde{I} = C\frac{d\tilde{V}}{dt} \tag{168}$$

$$\tilde{I} = j\omega C V_s e^{j\phi} \tag{169}$$

Looking closer at this equation, we find something interesting. $V_s e^{j\phi}$, which appears at the end of the expression, is the phasor for voltage. Thus, we may rewrite the above equation as:

$$\tilde{I} = j\omega C\tilde{V} \tag{170}$$

Bringing this equation back to our definition of impedance, $Z = \tilde{V}/\tilde{I}$, we rearrange and find:

$$\frac{1}{j\omega C} = \frac{\tilde{V}}{\tilde{I}} \tag{171}$$

Thus, the impedance of a capacitor, Z_c, is defined as:

$$\boxed{Z_c = \frac{1}{j\omega C}} \tag{172}$$

Just as we defined the impedance of a resistor and capacitor, we must also define the impedance of an inductor. We begin with the current-voltage relationship for an inductor:

$$\tilde{V} = L\frac{d\tilde{I}}{dt} \tag{173}$$

Now, we follow a very strategy very similar to that used to calculate the impedance of a capacitor. Representing the current through the inductor in complex form, we may find its derivative with respect to time:

$$I(t) = Re(I_L e^{j(\omega t+\phi)}) \tag{174}$$

$$I(t) = Re(I_L e^{j\phi} e^{j\omega t}) \tag{175}$$

Taking the time derivative, we find:

$$\frac{dI}{dt} = Re(I_L j\omega e^{j\phi} e^{j\omega t}) \tag{176}$$

Thus, the phasor of the derivative of current is:

$$\frac{d\tilde{I}}{dt} = j\omega I_L e^{j\phi} \tag{177}$$

Recognizing $I_L e^{j\phi}$ as the phasor for current, we then substitute $\frac{d\tilde{I}}{dt}$ back into our current-voltage relationship and write:

$$\tilde{V} = j\omega L\tilde{I} \tag{178}$$

Dividing by $\tilde{I}$, we find:

$$\frac{\tilde{V}}{\tilde{I}} = j\omega L \tag{179}$$

Since impedance is defined as $\tilde{V}/\tilde{I}$, we thus conclude that the impedance of an inductor, Z_L, is:

$$\boxed{Z_L = j\omega L} \tag{180}$$

Note the similarity in the forms of capacitor and inductor impedance! Also note that since to the formula for impedance, $Z = V/I$, has the same format as Ohm's Law, $R = V/I$, impedance has units of ohms (Ω) no matter which component is used.
In summary:

Component	*Impedance*	*Units*
Resistor	R	$Ohms(\Omega)$
Capacitor	$\frac{1}{j\omega C}$	$Ohms(\Omega)$
Inductor	$j\omega L$	$Ohms(\Omega)$

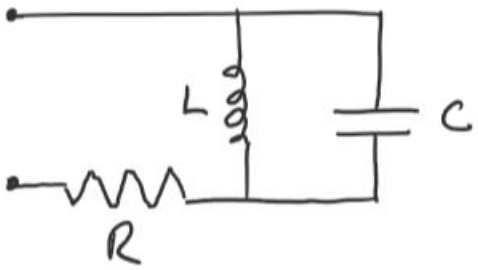
What does the definition of impedance mean for finding the **equivalent impedance** of groups of circuit elements?
Because all of the components are now governed by the same equation, $Z = \tilde{V}/\tilde{I}$, which is of the same form as Ohm's Law, we may add together groups of components (resistors, capacitors, and inductors) *just* as we would with resistors!
Thus, for a group of n components (resistors, inductors, and capacitors) in *series*, the equivalent impedance is:

$$\boxed{Z_{eq} = Z_1 + Z_2 + ... + Z_n} \tag{181}$$

And for n components in parallel, we may write:

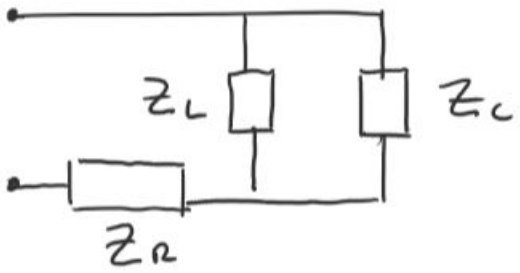
$$\boxed{\frac{1}{Z_{eq}} = \frac{1}{Z_1} + \frac{1}{Z_2} + ... + \frac{1}{Z_n}}$$

Let's now solve an equivalent impedance example problem.
Find the equivalent impedance of the components in the circuit below:



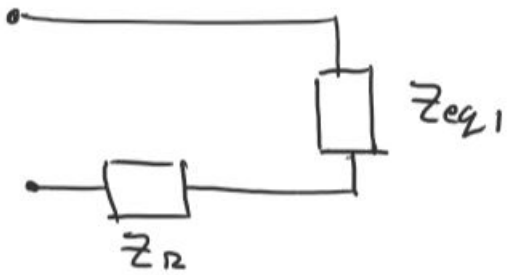
When finding the equivalent impedance of a group of components, we may use the following procedure:

1. **Redraw the circuit with boxes instead of components.** When working with AC circuits, it's convention to redraw each component as a simple box instead of the actual component drawing. This reinforces the idea that each component follows the same law for impedance, current, and voltage: $Z = \tilde{V}/\tilde{I}$. By replacing the components with boxes, we're indicating that we're now thinking of the components simply in terms of their impedances. After redrawing, we label each component.



Above: the redrawn circuit with components labeled as boxes

2. **Apply the equivalent impedance formulas until a single component remains.** Using the formulas for equivalent impedance, we first recognize that the components Z_L and Z_C are in *parallel.* Thus, we may use the parallel impedance formula to simplify the circuit to the following:



Where by the parallel impedance formula, Z_{eq1} is:

$$Z_{eq1} = \left(\frac{1}{Z_L} + \frac{1}{Z_C}\right)^{-1} = \frac{Z_L Z_C}{Z_L + Z_C} \tag{182}$$

To get the circuit to one equivalent component, we must reduce again. Looking at the circuit above, we realize that Z_{eq1} and Z_R are in *series.* Thus, we may use the formula for series impedance to simplify the circuit to a single equivalent component:

Using the series formula and substituting our value for Z_{eq1}:

$$Z_{eq} = Z_R + Z_{eq1} \tag{183}$$

$$Z_{eq} = Z_R + \frac{Z_L Z_C}{Z_L + Z_C} \tag{184}$$

3. **Substitute in the impedance of each component.** Using the formulas for the impedance of resistors, capacitors, and inductors, substitute in values to find the total equivalent impedance.

$$Z_{eq} = Z_R + \frac{Z_L Z_C}{Z_L + Z_C} \tag{185}$$

$$Z_{eq} = R + \frac{(\frac{1}{j\omega C})(j\omega L)}{(\frac{1}{j\omega C}) + (j\omega L)} \tag{186}$$

Using the laws of complex numbers, we simplify this expression to:

$$Z_{eq} = -\frac{-C\,L\,R\,\omega^2 + j\,\omega\,L + R}{C\,L\,\omega^2 - 1} \tag{187}$$

Thus, the problem is complete![8]

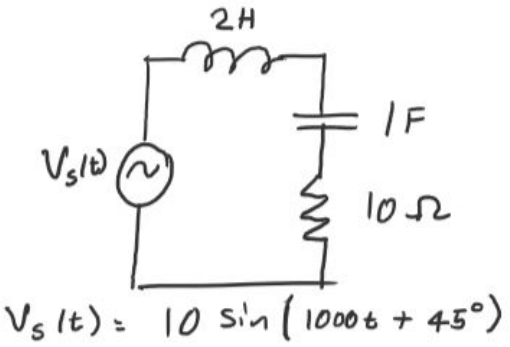
5.4 Solving Circuits with Phasors

Now that we've defined the necessary theory for phasors and have discussed the important concept of impedance, we're ready to solve AC circuits using phasors. At this point, you may be asking yourself - so what? At the moment, phasors may seem like an overly complex way of solving problems with AC circuits. From complex numbers to sometimes challenging identities, at first sight they may seem to be more trouble than they're worth.

However, as we'll soon see, phasors make the analysis of AC circuits almost as simple as our previous DC analyses.

The main reason for this is that with phasor analysis, *all* of our past circuit principles, KCL, KVL, equivalence, and superposition, *still* apply! This incredibly useful result means that we may solve AC circuits using methods almost *identical* to those we've developed thus far.

Let's now illustrate the process of **phasor analysis** with the following example: *Solve for the steady state voltage across the* 10Ω *resistor:*[9]



[8] Problems such as these can be greatly simplified with the use of the MATLAB Symbolic Library, which can simplify complex number expressions.

[9] In Phasor Analysis, we solve for the **steady state** voltage and current - that is, voltage and current after the initial "unstable" transient response has died out.

Note that when we have an AC source, the voltage source is drawn with a $\sim$ inside rather than a $\pm$. Let's now develop a procedure for solving AC circuits.

1. **Convert all sources to standard AC form.** Recall that the standard AC form for a source is:

$$V(t) = V_s \cos(\omega t + \phi) \tag{188}$$

Using this form, we developed the theory for phasor analysis. Thus, we *must* ensure that all of our sources are written in this way. Looking at our source V_s, we realize that it is *not* yet in standard form. To convert, we use the relationship:

$$\sin\theta = \cos(\theta - 90^\circ) \tag{189}$$

Using this formula, we find:

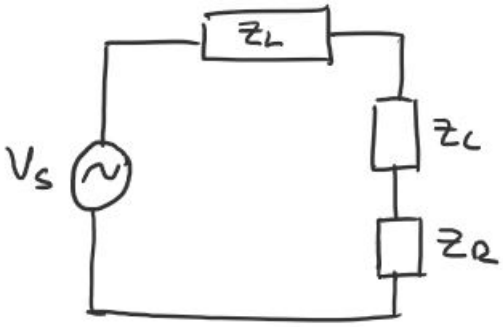
$$V_s(t) = 10\cos(1000t - 45^\circ) \tag{190}$$

2. **Convert all sources to phasor form.** Now that all of our sources are in standard AC form, we may convert them into *phasor* form. Here, we only have one source that we need to convert. Let's use the exponential form to represent our voltage source.

$$\tilde{V}_s = V_s e^{j\phi} \tag{191}$$

$$\tilde{V}_s = 10e^{-j45^\circ} \tag{192}$$

3. **Redraw the circuit with boxes for each component. Find the impedance of each component.** Recall that when working with phasors, we redraw each component as a box to emphasize that we only care about the impedance. Redrawing the provided circuit, we find:



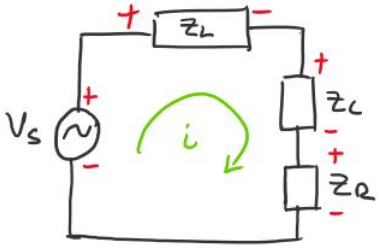
Now, we define the impedance for each component:

$$Z_L = j\omega L = j(1000)(2) = 2000j\ \Omega \tag{193}$$

$$Z_C = \frac{1}{j\omega C} = \frac{1}{1000j}\ \Omega \tag{194}$$

$$Z_R = R = 10\ \Omega \tag{195}$$

4. **Use Kirchoff's Laws to write equations for the circuit.** At this point in the analysis, we may solve the circuit just like any DC circuit! Let's write a KVL expression for our circuit. Note that we still assume a direction for current and use the same sign convention as DC circuits.



Using our sign convention, we write out the following KVL equation:

$$V_{Z_L} + V_{Z_C} + V_{Z_R} - \tilde{V}_s = 0 \tag{196}$$

5. **Apply the phasor current-voltage relationship. Solve for current.** Now that we have our KVL expression, we may apply the phasor current-voltage relationship $Z = \tilde{V}/\tilde{I}$ (just like Ohm's Law!) to develop a solvable equation. We thus write:

$$\tilde{I}Z_L + \tilde{I}Z_C + \tilde{I}Z_R - \tilde{V}_s = 0 \tag{197}$$

Now, we solve this equation for $\tilde{I}$:

$$\tilde{I}Z_L + \tilde{I}Z_C + \tilde{I}Z_R = \tilde{V}_s \tag{198}$$

$$(Z_L + Z_C + Z_R)\tilde{I} = \tilde{V}_s \tag{199}$$

$$\frac{\tilde{V}_s}{Z_L + Z_C + Z_R} = \tilde{I} \tag{200}$$

Note: *don't* substitute values for $\tilde{V}_s$ or impedance until the end of the problem. This way, we can avoid complex number arithmetic until the last step.

6. **Substitute back into the I-V relationship.** To solve for the voltage across the resistor, all that remains is to substitute our formula for $\tilde{I}$ back into the current-voltage relationship, $Z = \tilde{V}/\tilde{I}$.

$$\tilde{V}_R = \tilde{I}_R Z_R \tag{201}$$

Substituting for $\tilde{I}$, we find:

$$\tilde{V}_R = \frac{\tilde{V}_s Z_R}{Z_L + Z_C + Z_R} \tag{202}$$

7. **Substitute impedance and source values. Solve for voltage in complex form.** Now that we have an expression for voltage across the resistor, we first plug in our impedances.

$$\tilde{V}_R = \frac{10\tilde{V}_s}{2000j + \frac{1}{1000j} + 10} \tag{203}$$

Now, to find a useful formula for $\tilde{V}_R$, we must convert our phasor for $\tilde{V}_s$ into *complex form* before plugging it in. Using the formula for complex form $(a + bj)$, we find:

$$\tilde{V}_s = a + bj \tag{204}$$

$$|\tilde{V}_s| = 10 \tag{205}$$

$$a = |\tilde{V}_s| \cos\phi = 10\cos(-45) = \frac{10}{\sqrt{2}} \tag{206}$$

$$b = |\tilde{V}_s| \sin\phi = 10\sin(-45) = -\frac{10}{\sqrt{2}} \tag{207}$$

$$\tilde{V}_s = \frac{10}{\sqrt{2}}(1 - j) \tag{208}$$

Now that we have $\tilde{V}_s$ in complex form, we may plug it into the expression for $\tilde{V}_R$.

$$\tilde{V}_R = \frac{10\tilde{V}_s}{2000j + \frac{1}{1000j} + 10} \tag{209}$$

$$\tilde{V}_R = \frac{100(1 - j)}{\sqrt{2}(2000j + \frac{1}{1000j} + 10)} \tag{210}$$

Now, using complex arithmetic (or a program such as MATLAB) to simplify, we find:

$$\tilde{V}_R = -0.0352 - 0.0355j \tag{211}$$

8. **Convert the answer to a function of time** Now that we have the voltage in *complex form*, we must convert it back to *phasor form* and then convert it to a function of time.
We begin by converting it to phasor form. We can do this by finding the *magnitude* and *phase* of the complex number. [10]

$$|\tilde{V}_R| = \sqrt{a^2 + b^2} = 0.05 \tag{212}$$

$$\phi = \tan^{-1}(b/a) = -135^{\circ} \tag{213}$$

[10] The MATLAB functions abs() and angle() respectively return the magnitude and phase of a complex number.

Thus, the phasor for voltage is:

$$\tilde{V}_R = 0.05e^{-j135^\circ} \tag{214}$$

Since we know frequency is unaffected by our linear circuit components, we may thus write this as a function of time using the frequency of the voltage source ($\omega = 1000$). Thus, the voltage across the resistor as a function of time is:

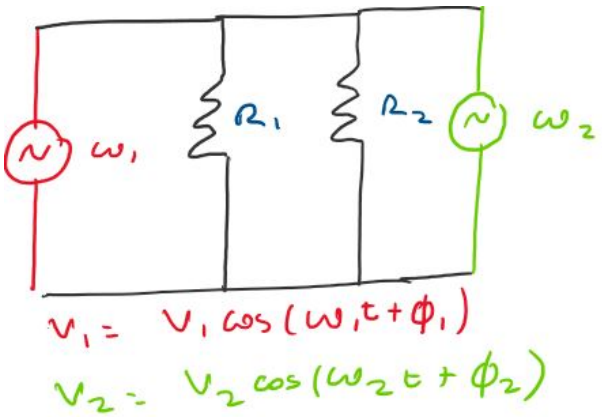
$$\boxed{V_R(t) = 0.05\cos\left(1000t - 135^\circ\right)} \tag{215}$$

And our solution is completed!

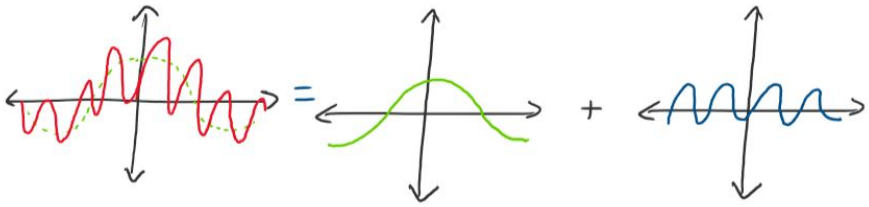
As you can see, the process for solving AC circuits is *very* similar to that for solving DC circuits! The only difference is: we have a few extra steps at the beginning and end to process our complex numbers.
Looking back on the problem we've just solved, you may be able to appreciate some of the magic that's just happened! By solving the circuit with phasor analysis, we were able to avoid differential equations *entirely* and find a purely algebraic solution.

5.5 Generalizing Phasor Analysis

In our example and discussion of phasors so far, we've encountered problems with a single source operating at a certain frequency. What happens if instead of a single frequency, we have multiple sources, each with their own frequencies, to deal with?



To solve circuits such as the one above, with two or more frequencies, we may use superposition to analyze the circuit *one frequency at a time.* At the end, as we do in a normal superposition analysis, we simply add the resulting signals up!
We can also apply this idea of adding multiple frequencies if we have a highly complex signal such as the one below an an input to the circuit.



Here, we have a function that's composed of two sinusoids of different frequencies. Although the resulting signal looks complex, we may apply the same idea of analyzing each frequency separately and adding the results.
This idea of adding sinusoids of different frequencies to produce a more complex signal is detailed in Fourier's Theorem, which states that an arbitrary periodic signal can be expressed as a sum of sinusoids.

$$V(t) = \frac{V_0}{2} + \sum_{n=1}^{N} A_n \cos\left(\omega_n t + \phi_n\right) \tag{216}$$

Because we're able to express any repeating signal as a sum of cosines, phasor analysis may be applied to *any* repeating signal.
Although the details of breaking up signals of multiple frequencies are out of the scope of ME 100, thinking of some of the complex signals you can make is an interesting exercise![11]

[11] You can learn more about the analysis of signals in courses such as ME 132 (dynamic systems & feedback control), EE 120 (signals & systems), and Math 118 (fourier analysis, wavelets, and signal processing).

6 Frequency Response

In our most recent studies in electronics, we learned a new method to analyze time-varying signals. Recall that instead of using our typical notion of an AC signal, a sinusoidal[12] function that changes in time, we opted to use complex numbers to represent the the inputs to our circuits.
As we saw, this not only simplified the style of analysis, through eliminating the need for differential equations, but also enabled us to use circuit analysis techniques very similar to those for DC circuits.
Now that we've introduced the fundamentals of phasor analysis, we look to gain even more insight into the world of time-varying signals. In this chapter, we provide a more in-depth discussion of the response of circuits to sinusoidal signals, known as **frequency response**.

6.1 Phasor Operations

Before we continue with our analysis of AC circuits, it's important to develop several further properties of phasors that will help us at a later stage in our analysis.
One operation that will often come up in our problems is **phasor division**. If we have two voltage phasors, for example, $\tilde{V}_1$ and $\tilde{V}_2$, how may we conveniently calculate $\tilde{V}_1/\tilde{V}_2$? To answer this question, we turn to the *exponential form* for a phasor. Recall that in exponential form, a phasor may be represented as:

$$\tilde{V} = V_s e^{j\phi} \tag{217}$$

Now, let's represent our two signals, $\tilde{V}_1$ and $\tilde{V}_2$, in this form. We may express them as:

$$\tilde{V}_1 = V_1 e^{j\phi_1} \tag{218}$$

$$\tilde{V}_2 = V_2 e^{j\phi_2} \tag{219}$$

Now, using the division rule for the exponential function, $e^a/e^b = e^{a-b}$, we may divide $\tilde{V}_1$ by $\tilde{V}_2$ as follows:

$$\frac{\tilde{V}_1}{\tilde{V}_2} = \frac{V_1 e^{j\phi_1}}{V_2 e^{j\phi_2}} \tag{220}$$

$$\frac{\tilde{V}_1}{\tilde{V}_2} = \frac{V_1}{V_2} e^{j\phi_1 - j\phi_2} \tag{221}$$

$$\frac{\tilde{V}_1}{\tilde{V}_2} = \frac{V_1}{V_2} e^{j(\phi_1 - \phi_2)} \tag{222}$$

This tells us that dividing two phasors gives us a new phasor with a magnitude equal to the quotient of the magnitudes of the two phasors and a phase equal

[12] Note: both sine and cosine are "sinusoidal" functions.

to the difference in the phases of the two phasors.
Expressing this result in standard phasor form, if we have two phasors, $\tilde{V_1} = V_1\angle\phi_1$ and $\tilde{V_2} = V_2\angle\phi_2$, using the result from the exponential form, we know:

$$\frac{\tilde{V_1}}{\tilde{V_2}} = \frac{V_1}{V_2}\angle\phi_1 - \phi_2 \tag{223}$$

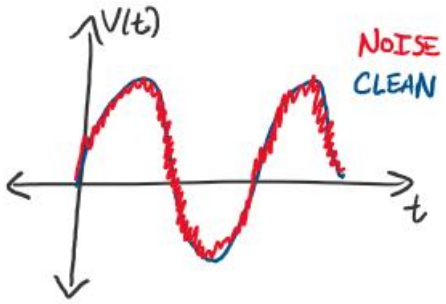
Just as we defined a formula for phasor division, we can also define a formula for phasor multiplication. Using a similar derivation (start at the exponential form and use exponent laws), we find that the product of two phasors is as follows:

$$\tilde{V_1} \cdot \tilde{V_2} = V_1V_2\angle\phi_1 + \phi_2 \tag{224}$$

As we continue to develop our understanding of the frequency response of circuits, we'll keep these two useful operations in mind.

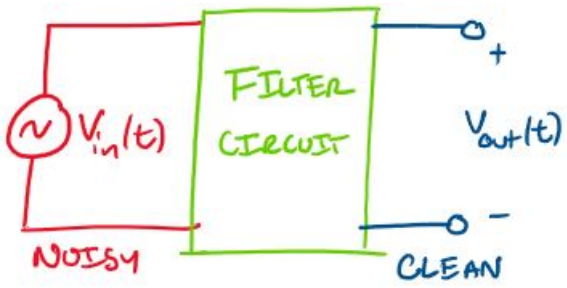
6.2 Filtering

As you've no doubt seen in your lab sections, electrical **noise** is a significant challenge to overcome in circuits.
Whereas in theory, signals are perfectly smooth and regular, in practice, we often end up with unintended background noise that interrupts our signal. Noise may be thought of as an unintentional signal that distorts our desired signal.



Above: A red signal with high frequency noise, compared to the desired clean blue signal beneath.

Let's consider a practical example of noise: listening to music through headphones. If the headphones aren't of a particularly high quality, you might often hear a hum or whine in the background in addition to the musical signal you want to hear. This is another example of undesired noise.
In electronics, we're often faced with the intimidating challenge of *eliminating* noise. To reduce the effects of noise in our circuits, we design special circuits known as electrical **filters**.
Let's now set up and motivate the problem statement for an electrical filter. We may model a filter with the following diagram:



On the left side, we have a noisy voltage signal, V_{in}, which we'd like to filter. We'd like this noisy voltage signal to enter our specially designed filter circuit and come out on the other side as V_{out}, the clean signal that we desire.

6.2.1 Transfer Functions

Before we dive into the details of *how* we design these filter circuits, it's important to develop several concepts that are key to filtering. The first of these is the transfer function.

Notice that in the circuit above, our entire circuit problem revolves around two quantities: V_{out} and V_{in}. The **transfer function** for the circuit is the ratio of V_{out} to V_{in} in *phasor form*. Because our transfer function is in phasor form, it will be a function of frequency, ω. This is because our filter circuit, which may contain capacitors and inductors with ω dependent impedance, will impact the ratio of V_{out} and V_{in}.

Mathematically, using $H(\omega)$ as the name of our transfer function, we express the transfer function as:

$$\boxed{H(\omega) = \frac{\tilde{V}_{out}}{\tilde{V}_{in}}} \tag{225}$$

From this form, we can see that a transfer function is simply the quotient of two phasors. Therefore, to find the magnitude and phase of the transfer function, we can use the phasor division relationships we just derived.

To find the **magnitude** of a transfer function, we divide the magnitude of the numerator by the magnitude of the denominator.

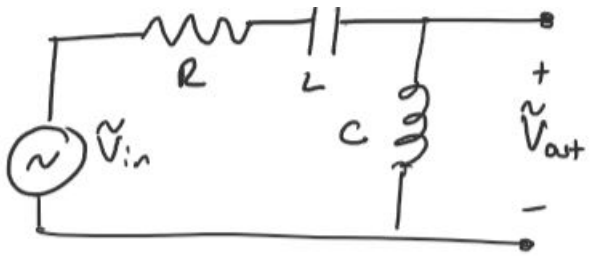
$$|H(\omega)| = \frac{|\tilde{V}_{out}|}{|\tilde{V}_{in}|} \tag{226}$$

To find the **phase** of a transfer function, using the formula for dividing two phasors, we find that the phase is the phase of the numerator minus the phase of the denominator.

$$\angle H(\omega) = \angle\tilde{V}_{out} - \angle\tilde{V}_{in} \tag{227}$$

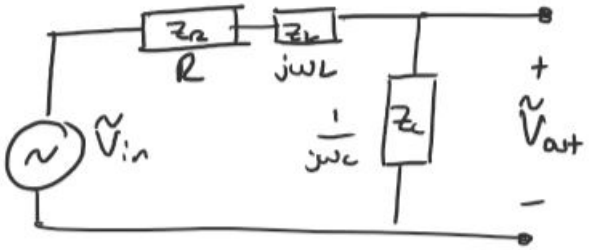
Let's practice finding a transfer function with the following example.

Find the transfer function $H(\omega) = \tilde{V}_{out}/\tilde{V}_{in}$ for the following circuit:



When finding the transfer function, we may use the following procedure:

1. **Convert the circuit to phasor form.** Recall that when applying phasor analysis to a circuit, we begin with two things: first, we redraw the circuit with boxes for components. Second, find the impedance of each component. Completing these two steps, the circuit will now look like the following:



2. **Solve for $\tilde{\mathbf{V}}_{\mathbf{out}}$ as a function of $\tilde{\mathbf{V}}_{\mathbf{in}}$.** Now that we've redrawn our circuit and prepared it for phasor analysis, we may use our circuit analysis techniques to solve for $\tilde{V}_{out}$ in terms of the other circuit variables.
Let's begin by writing a KVL equation around the loop in the circuit and solving for $\tilde{I}$, the phasor for current. This will help us in finding $\tilde{V}_{out}$.

$$V_R + V_L + V_C - \tilde{V}_{in} = 0 \tag{228}$$
$$Z_R\tilde{I} + Z_L\tilde{I} + Z_C\tilde{I} = \tilde{V}_{in} \tag{229}$$
$$(Z_R + Z_L + Z_C)\tilde{I} = \tilde{V}_{in} \tag{230}$$
$$\frac{\tilde{V}_{in}}{Z_R + Z_L + Z_C} = \tilde{I} \tag{231}$$

Now that we have $\tilde{I}$, we may solve for the voltage drop across the capacitor. Looking closely at the circuit, we find that this equals $\tilde{V}_{out}$. Let's plug into the impedance current-voltage relationship and solve.

$$\tilde{V}_{out} = \tilde{V}_C = \tilde{I}Z_C \tag{232}$$
$$\tilde{V}_{out} = \frac{\tilde{V}_{in}Z_C}{Z_R + Z_L + Z_C} \tag{233}$$

3. **Solve for $\tilde{\mathbf{V}}_{\mathbf{out}}/\tilde{\mathbf{V}}_{\mathbf{in}}$.** Now that we have an equation for $\tilde{V}_{out}$, we simply rearrange to find $\tilde{V}_{out}/\tilde{V}_{in}$.

$$\frac{\tilde{V}_{out}}{\tilde{V}_{in}} = \frac{Z_C}{Z_R + Z_L + Z_C}$$

4. **Substitute in impedance values**. Now that we have all we need to solve for the transfer function, all that remains is to substitute values for the impedance of each component.

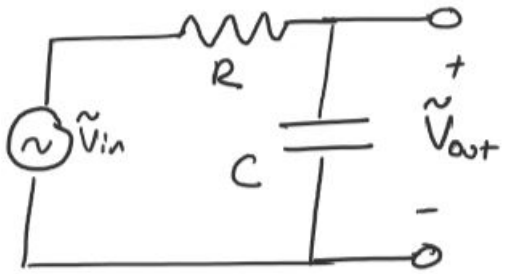
$$H(\omega) = \frac{\tilde{V}_{out}}{\tilde{V}_{in}} = \frac{Z_C}{Z_R + Z_L + Z_C} \tag{234}$$

$$H(\omega) = \frac{\frac{1}{j\omega C}}{R + j\omega L + \frac{1}{j\omega C}} \tag{235}$$

We now have our final transfer function as a function of frequency, ω.

6.3 The Low Pass Filter

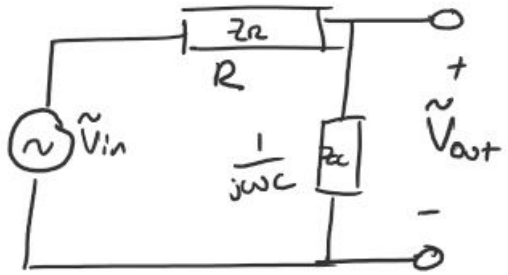
Now that we've developed an appropriate background in transfer functions, let's turn our attention back to filtering. Consider the following filtering problem. Imagine we're trying to measure a low frequency signal, but we're encountering a significant amount of high frequency noise. How can we design a *filter* to preserve our desired *low* frequency signal and block out the *high* frequency noise?
The following circuit, called a **first order low pass filter**, is designed to do just this. A low pass filter **passes** low frequency signals, and **rejects** high frequency signals. Note that the term "first order" comes from the fact that a first order differential equation can describe the circuit.



Above: A first order low pass filter. By adjusting the values of R and C, we can select which frequency signal we'd like to block out.

What exactly do we mean by pass and reject? When a signal *passes* through a filter, it comes out of the filter with a phase and magnitude very close to the original. When a signal is *blocked* by a filter, it comes out of the filter with a magnitude close to *zero*.

Let's now solve for the transfer function of the low pass filter circuit, and see what information it can tell us about *how* the low pass filter actually works. First, let's redraw the circuit as a phasor circuit with impedances.[13]



Using this circuit, we may write the KVL equation around the loop to find:

$$Z_R\tilde{I} + Z_C\tilde{I} - \tilde{V}_{in} = 0 \tag{236}$$

To find the transfer function, we may begin by solving for current.

$$\tilde{I} = \frac{\tilde{V}_{in}}{Z_R + Z_C} \tag{237}$$

Now, to solve for $\tilde{V}_{out}$, we recognize that $\tilde{V}_{out}$ is the same as $\tilde{V}_C$, as they're measured across the same nodes. Thus, applying the phasor $I-V$ relationship to the capacitor, we find:

$$\tilde{V}_{out} = \tilde{V}_C = \tilde{I}Z_C \tag{238}$$

$$\tilde{V}_{out} = \frac{\tilde{V}_{in}Z_C}{Z_R + Z_C} \tag{239}$$

Now, we divide both sides by $\tilde{V}_{in}$ to get our transfer function, $H(\omega) = \tilde{V}_{out}/\tilde{V}_{in}$.

$$H(\omega) = \frac{\tilde{V}_{out}}{\tilde{V}_{in}} = \frac{Z_C}{Z_R + Z_C} \tag{240}$$

Substituting in our values for impedance and simplifying, we arrive at our formula for the transfer function as follows:

$$H(\omega) = \frac{\frac{1}{j\omega C}}{R + \frac{1}{j\omega C}} \tag{241}$$

$$H(\omega) = \frac{1}{j\omega RC + 1} \tag{242}$$

Let's now find the magnitude and phase of this transfer function, and see what useful information we can extract!

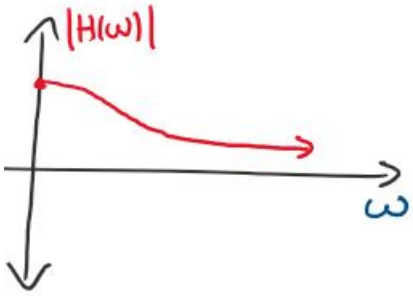
[13] Notice that the low pass filter circuit is actually a voltage divider! You may use the voltage divider equations to speed up your analysis instead of starting from KCL and KVL.

Recall, the magnitude of a transfer function is equal to the magnitude of the numerator divided by the magnitude of the denominator. Using the complex magnitude formula, we arrive at the following result:

$$|H(\omega)| = \frac{|1|}{|j\omega RC + 1|} \tag{243}$$

$$|H(\omega)| = \frac{1}{\sqrt{\omega^2 R^2 C^2 + 1}} \tag{244}$$

As we can see from this formula, if we plug in a very low frequency for ω, the magnitude of our transfer function will be *very* close to 1. This means that the magnitudes of *low frequency* signals will be preserved by the low pass filter. Alternatively, if we plug in a high frequency, we find that the magnitude of the transfer function approaches 0 - thus, high frequencies are *rejected* by the low pass filter!
Plotting magnitude versus frequency on a graph, we produce the following plot:



Above: The magnitude vs. frequency plot for the low pass filter. Low frequencies are preserved, while high frequencies approach 0.

Let's now carry out the same analysis for phase. If our low pass filter works, then the phase shift for low frequency signals should be very close to zero. Recalling that the phase of a transfer function is the phase of the numerator minus the phase of the denominator:

$$\angle H(\omega) = \angle 1 - \angle(j\omega RC + 1) \tag{245}$$

$$\tag{246}$$

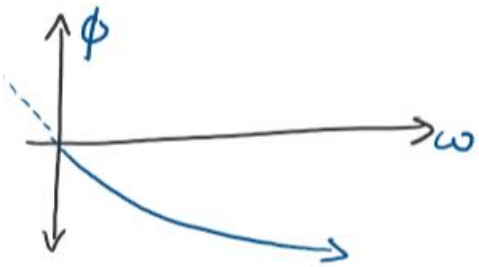
Applying the formula for the phase of a complex number:

$$\angle H(\omega) = 0 - \tan^{-1}(\omega RC) \tag{247}$$

$$\angle H(\omega) = -\tan^{-1}(\omega RC) \tag{248}$$

Plotting this result, we observe that once again, low frequencies are preserved, and have only a small phase shift from their original.[14]

[14] Note that since we're not concerned with preserving the high-frequency signals, we're not concerned with their phase.



Above: The phase plot for a low pass filter. Note that low frequencies have only a small phase shift.

Thus, our low pass filter is successful!

6.4 Decibels and Bode Plots

You may have noticed in the graphs of magnitude and phase versus frequency that we were only able to describe a limited range of frequencies and magnitudes. Since noise can often be of a very high frequency compared to our desired system, we want to *expand* what we're able to fit on our graph to include a wider range of frequencies and magnitudes. How can we do this?
First, we define a new unit of magnitude, called the **decibel** (dB). The decibel to magnitude conversion ratio is defined as follows:

$$|H(\omega)|_{dB} = 20\log(|H(\omega)|) \tag{249}$$

In words, the magnitude in decibels is equal to 20 times the log (base 10) of the magnitude of the transfer function. This log scale will help us display a much wider range of magnitudes, corresponding to a wider range of frequencies, in the same amount of space.
Let's now practice converting magnitudes to decibels to get a feel for this new unit. It's important to pay close attention to the sign in the following examples.
Convert the following to decibels:

$$|H_1(\omega)| = 100 \tag{250}$$
$$|H_2(\omega)| = 0.001 \tag{251}$$

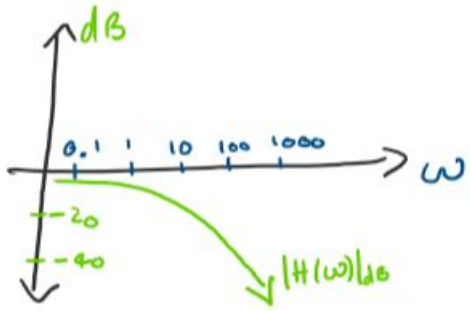
To convert to decibels, we simply plug into the formula defined above. For the first magnitude:

$$|H_1(\omega)|_{dB} = 20\log(100) \tag{252}$$
$$|H_1(\omega)|_{dB} = 20 \cdot 2 \tag{253}$$
$$|H_1(\omega)|_{dB} = 40\ dB \tag{254}$$

Now, let's try converting the second value to decibels.

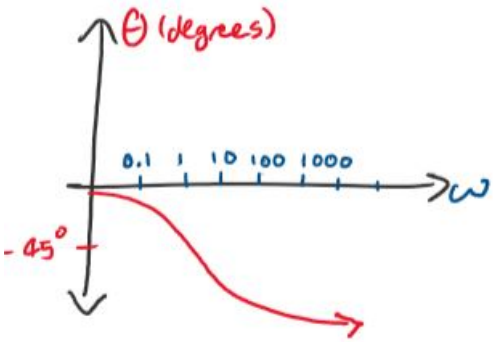
$$|H_2(\omega)|_{dB} = 20\log(0.001) \tag{255}$$
$$|H_2(\omega)|_{dB} = 20 \cdot -3 \tag{256}$$
$$|H_2(\omega)|_{dB} = -60\ dB \tag{257}$$

Note from the second example that converting a decimal magnitude to decibels results in a *negative* number, a result that will be particularly important when evaluating graphs with units of decibels.
What else can we do to fit in more information on our graphs of frequency vs. magnitude? Because we defined a log scale for magnitude, it's only natural to define a log scale for frequency as well. Now, instead of representing a constant increment, each increment of frequency on the graph will grow by a factor of ten.
Let's plot the magnitude in decibels against frequency on a log scale and see what our graph looks like. We observe the following plot:



Above: Notice how the log scale lets us describe much higher frequencies

This plot is known as the **magnitude Bode Plot** for a low pass filter. A magnitude Bode plot is a plot of the magnitude in decibels vs. frequency on a log scale.
Similarly, the **phase Bode Plot** (below) is a plot of phase (in degrees or radians) versus frequency on a log scale.



These two plots are crucial tools in analyzing the filters we define. Looking closely at the magnitude Bode plot for our low pass filter, for example, we observe that for low frequencies, the magnitude is unaffected, while for high frequencies, the magnitude quickly becomes a large negative decibel (a very small decimal magnitude).
We made read our phase plot in a similar way, finding that for low frequencies,

the phase shift from the filter is very small, while at high frequencies, the phase shift approaches $-90°$.
Bode plots are an essential tool in evaluating the function and effectiveness of electrical filters.

6.4.1 Corner Frequency

How can we tell when our low frequency ends and our high frequency begins? To have a general sense of whether our signal will be rejected by the filter or not, we define a special point on the Bode plot known as the **corner frequency** (also known as the cutoff or break frequency). This is defined as the frequency that makes our transfer function equal to the following:

$$H(\omega_b) = \frac{1}{1+j} \tag{258}$$

Looking at our low pass filter transfer function, we observe that ω_b for the low pass filter is equal to $1/RC$, or $1/\tau$, for τ the RC time constant.
Let's now apply the magnitude formula to identify the magnitude of the transfer function at the corner frequency.

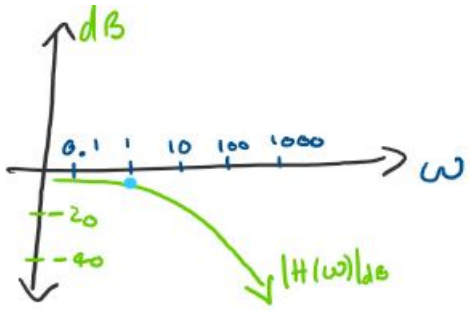
$$|H(\omega_b)| = \frac{1}{\sqrt{1+1}} \tag{259}$$

$$|H(\omega_b)| = \frac{1}{\sqrt{2}} \approx 0.707 \tag{260}$$

Converting this to decimals, we find:

$$|H(\omega_b)|_{dB} = -3\ dB \tag{261}$$

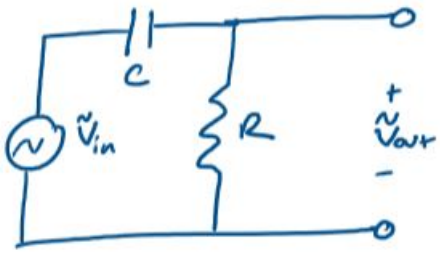
Oftentimes, when presented with a Bode plot, instead of frequency being in units of radians/second, it may be presented in multiples of the corner frequency. On a Bode Plot, you may recognize the corner frequency as the frequency at which the magnitude plot begins to make its change from flat to angled.



Above: the corner frequency for an example low pass filter, plotted in light blue. For this filter, $\omega_b \approx 1$.

6.5 The High Pass Filter

Thus far, we've only considered one type of filter: the low pass filter. If we're able to design a filter to filter out high frequency signals and pass low frequency signals, we should logically be able to do the opposite as well.
The **high pass filter** is a filter that preserves high frequency signals and blocks out low frequency signals. The circuit for the high pass filter is below:



Above: the circuit for a high pass filter

Notice that the high pass filter circuit is interestingly just the low pass filter with the resistor and capacitor switched!
As with the low pass filter, the high pass filter transfer function and Bode Plots may be solved for.
Note that the above is *not* the only layout for a high pass filter, nor was the low pass filter described earlier unique in its design. Rather, the high and low pass filters presented here are two of the most common.
It's a great exercise to discover what other circuits might act as high and low pass filters! Consider in particular the effects an inductor might have in the place of a capacitor.

6.6 Frequency Response in MATLAB

In this section of the course, we've delved deeper into the *frequency response* of circuits - the response of circuits to sinusoidal inputs. Now, we provide a brief introduction to some programming tools that can aid in our understanding of frequency response.
MATLAB is a powerful programming language that holds an exceptionally useful set of utilities for analyzing frequency response. Here, we describe procedures for using MATLAB to represent a transfer function and find its Bode plot.

6.6.1 Transfer Functions in MATLAB

Let's learn how to represent transfer functions in MATLAB. To follow along with the following tutorial, ensure you have the **Control System Toolbox** installed on your copy of MATLAB. If you don't have it installed or are unsure, you may install it by going to the "home" tab in the MATLAB editor and clicking "Add-Ons." You may then search for the toolbox in the window that pops up.

Let's use the low pass filter, which has the transfer function below, as our example. Let $R = 2\ \Omega$ and $C = 1\ F$.[15]

$$H(\omega) = \frac{1}{j\omega RC + 1} \tag{262}$$

$$H(\omega) = \frac{1}{2j\omega + 1} \tag{263}$$

Before we continue to enter this into MATLAB, let's make an important clarification. Where we use the product "$j\omega$" in our transfer function, MATLAB uses the variable "s" in its code. This is by convention of a more advanced signal analysis technique known as the *Laplace Transform*.
Let's rewrite our transfer function with s in the place of $j\omega$ to get it into the correct form. Simply replace all instances of $j\omega$ with the variable s as follows:

$$H(\omega) = \frac{1}{2s + 1} \tag{264}$$

We're now ready to enter our transfer function into MATLAB! To do this, we use the function **tf([numerator], [denominator])** as follows:

```
>> H_w = tf([1], [2, 1])
```

Note: we assign our transfer function to the variable H_w so we can refer to it later in the program.

The first argument of *tf* is an array of the coefficients of s in the numerator of our transfer function. Here, we only have 1 in the numerator, so we enter that. The second argument is an array of the coefficients of s in the denominator of the transfer function. We enter the coefficients in order of *descending* powers of s. The coefficient of s^1 comes first, and the coefficient of the constant (s^0) comes second. Hitting enter after typing the command, we see:

```
>> H_w = tf([1], [2, 1])

H_w =

     1
  -------
  2 s + 1

Continuous-time transfer function.
```

We now have a transfer function we can work with!

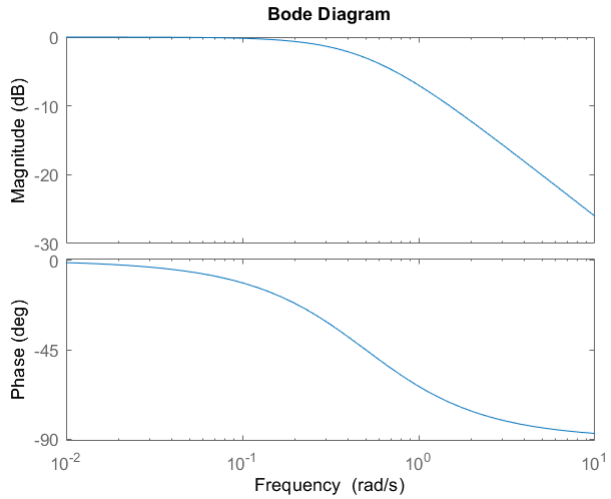
[15] 1 Farad is a *very* large capacitance in reality. Here, it's simply been chosen as an easy number to enter into MATLAB.

6.6.2 Bode Plots in MATLAB

Let's use our transfer function example from above to make a Bode plot for our low pass filter. To find the bode plot for a transfer function, simply use the function **bode(tf_name)**. Using our transfer function from above, we type:

```
>> bode(H_w)
```

Hitting enter, this command produces the following magnitude and phase Bode Plots:



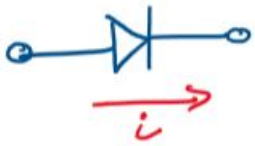
We've now successfully found the Bode Plots for a low pass filter!

7 Diodes and Transistors

Thus far in the course, we've discussed numerous methods of circuit analysis for a variety of sources. We've developed both AC and DC techniques, and have thoroughly explored the response of circuits to sinusoidal, AC signals.
In this section, we shift our focus to two new devices: the diode and the transistor. As we'll soon see, these components are essential to the controlling and amplifying of electrical signals in circuits. We'll soon come to appreciate them as powerful and useful tools in circuit design.

7.1 Diodes

So far in our circuit analysis, we've worked with currents that flow in directions purely determined by the positions of sources and circuit elements. What if we had another component that helped us *control* the direction in which current can flow?
The **diode** does exactly this. A diode, pictured below, is a circuit element that only allows current to flow through it in the direction of the arrow.



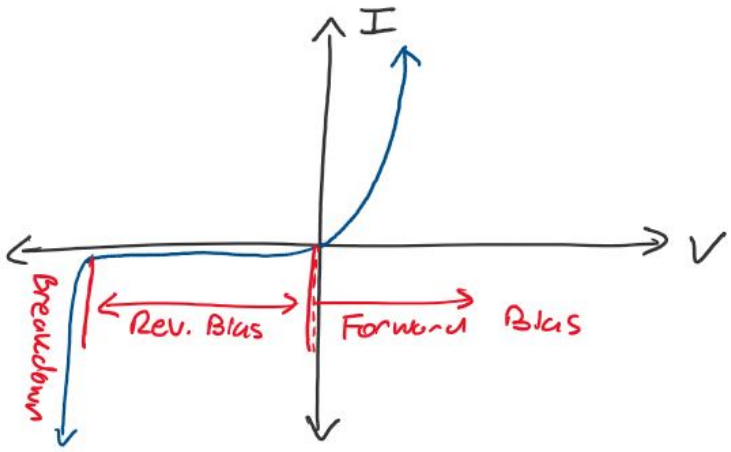
Above: current can only flow through a diode in the direction of the arrow

Let's quickly think back to our fluid analogy from the beginning of the semester. Recall that instead of flowing electrons, we thought of current as a flowing stream of water. In this analogy, the diode is a "check valve," a valve that only allows water to flow in one direction.

7.1.1 The Diode Current-Voltage Relationship

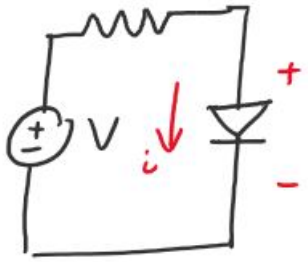
Let's think for a moment about how our electronic components have functioned thus far. We've found that with our components, from resistors to capacitors and beyond, there has been a *voltage drop* across the component - that is, as current passes through the component, voltage goes down.
With resistors, we were able to *quantify* this voltage drop via Ohm's Law, the linear equation $V = IR$, where we related the voltage drop across the resistor to the current passing through it and its resistance.
When working with capacitors and inductors, we defined similarly linear relationships between voltage, current, and a fundamental constant for the component. How can we do this for the diode?
As it happens, the diode is a *nonlinear* component, meaning that we can't describe its behavior with a simple linear equation like $V = IR$. Thinking back to our description of *load line analysis*, however, we recall that we can describe

nonlinear components in a useful way using graphs.
Below, we present the current-voltage relationship graph for a diode.



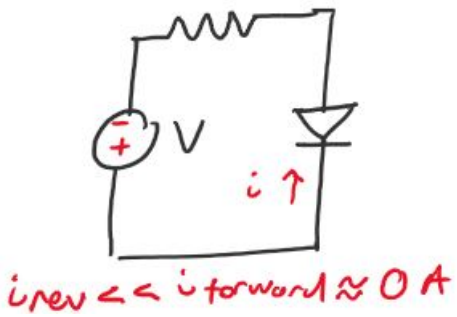
Above: the nonlinear current-voltage relationship for a diode

Let's break down what's happening in this graph - there's a lot of interesting information to digest!
For positive, *forward* voltages, we find that we get very *large* currents for *small* voltages. This is known as the **forward bias** region of the diode, and is where most diodes are designed to operate.



Above: a diode operating in the forward bias region

When a diode has a positive voltage being applied across it, because the current grows so quickly for such a small voltage, we often approximate a diode in forward bias as a wire. We'll come back to this assumption later in our analysis of diode circuits.
When voltage across the diode is negative, the diode operates in the **reverse bias** region. When a diode is in reverse bias, current tries to flow through the diode in the wrong way. Due to the micro-level design of the diode, however, it's incredibly difficult for current to flow in the reverse direction. Because of this, we see that even for large negative voltages, an almost negligible current flows through the diode in reverse bias.



Above: only very small currents can pass through the diode in reverse bias

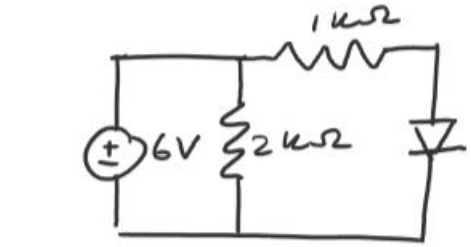
What happens, however, if the voltage we apply to the diode in reverse is *very* high? If the voltage exceeds a value known as as the **breakdown voltage**, the diode will enter the **breakdown** region. Once the breakdown voltage is exceeded, current flows *rapidly* through the diode.
Does operation in breakdown damage the diode? As it happens, diodes may operate in the breakdown region without trouble - the breakdown region is simply challenging to reach due to the *very* high negative voltages required.
Some diodes, known as **zener diodes**, are actually designed to operate in the breakdown region.

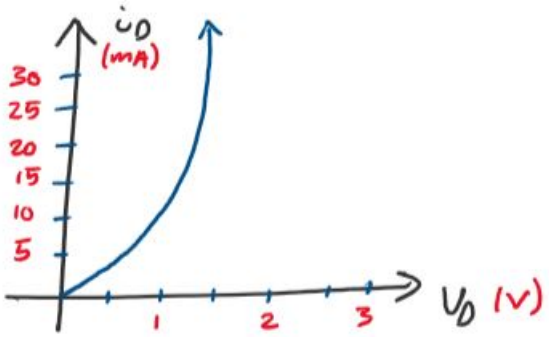
7.1.2 Analyzing Simple Diode Circuits

In the past, after defining the current-voltage relationship for a component, we were able to find the current and voltage passing through it when placed in a circuit with other components. Let's now do the same for a diode.
To analyze circuits with a single, nonlinear diode, we may use *load line analysis*. Recall the following procedure for load line analysis, which may be used in a circuit with a *single* nonlinear component.

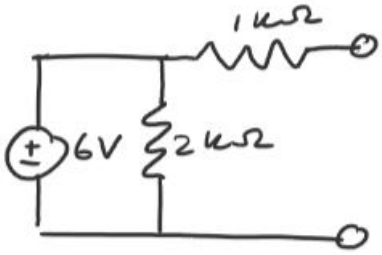
1. Find the Thevenin Equivalent around the nonlinear component.
2. Write a KVL equation using the Thevenin Equivalent.
3. Plot the KVL equation on the current-voltage graph for the component and find the intersection.

Let's now apply load line analysis to solve for the current and voltage across the diode in the following example:
Find the current and voltage across the nonlinear diode in the following circuit. Use the provided current-voltage graph to aid in your analysis.

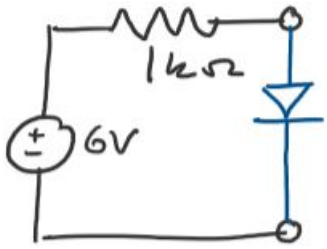




Let's follow the load-line analysis procedure to find the current and voltage for our diode. First, we begin by finding the Thevenin equivalent circuit around our diode. Representing the nodes at either end of our diode with two terminals, we find the Thevenin equivalent of the following circuit:



Using our Thevenin equivalent procedure to find the equivalent circuit of the above, we arrive at the circuit below and place our diode back between the terminals.



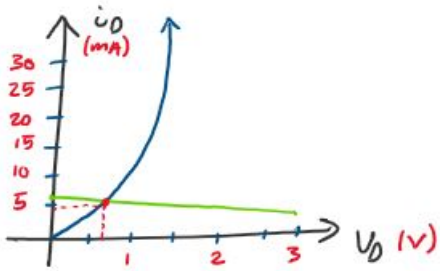
Now, we write a KVL equation around the loop of our Thevenin equivalent circuit. We then solve this equation for a relationship between voltage and current across the diode.

$$-V_s + iR + V_{diode} = 0 \tag{265}$$
$$-V_s + V_{diode} = -iR \tag{266}$$
$$\frac{V_s}{R} - \frac{V_{diode}}{R} = i \tag{267}$$

Substituting values and converting to units of mA, we get:

$$i = 6 - V_{diode} \tag{268}$$

Now, we simply plot this equation on our load line graph and find the intersection! This will be the operating point of the diode.

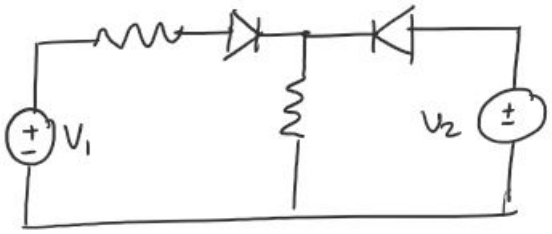


Above: the load line (green) and intersection point (red)

Thus, we conclude that the voltage across the diode will be approximately 0.6 V and the current passing through the diode will be approximately 5 mA. Our problem is now complete!

7.1.3 Ideal Diode Analysis

So far, we've discussed the analysis of *real* diodes - diodes with a nonlinear, graphical I-V relationship. As we saw in the previous problem, we were able to use load line analysis to reach a solution for a circuit in which we have one diode.
What happens, however, when we have multiple diodes, such as in the circuit below?

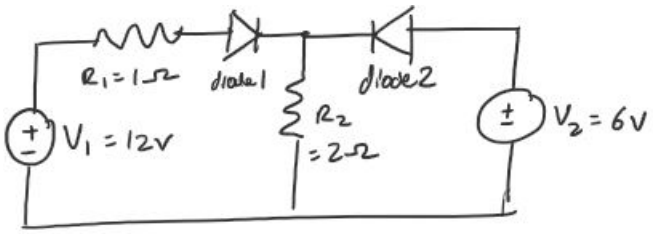


Since load line analysis only applies with circuits with a single nonlinear component, we must come up with a different method to solve these multi-diode circuits.
Instead of going through the whole process of solving a load-line analysis problem, in multi-diode circuits, we make a new assumption: that the diodes are **ideal**.
An ideal diode is a diode that has the following properties:

1. In the forward bias direction (in the direction of the arrow), the diode acts like a wire. It has zero voltage drop across it and may carry any current.

2. In the reverse bias direction (opposite to the arrow), the diode acts as an open circuit. It may have any voltage applied across it yet zero current will flow through it.

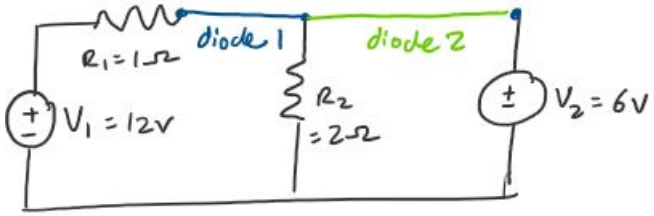
This assumption will greatly simplify our analysis, and will allow us to analyze a circuit with *any* number of diodes.
Before we begin our circuit analysis, however, let's take a moment to define a few terms that'll help us organize our ideas.
When a diode has current flowing through it in the *positive* direction, we say that the diode is **on**. When the diode has a *negative voltage* going across its terminals, and thus zero current passing through it, we say the diode is **off**.
With these terms in mind, let's now define a procedure for solving circuits with ideal diodes. Let's use the following example:
Find the currents going through ideal diodes 1 and 2.



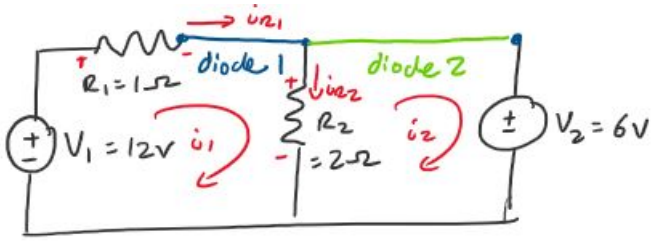
When solving circuits with multiple diodes, we use the following procedure:

1. **Assume a state (on or off) for each diode**. To begin, we'll assume, using our intuition for the circuit or a random guess, whether each diode is *on* or *off*. Remember, an ideal diode that's *on* acts like a wire, while an ideal diode that's *off* acts like an open circuit.
   For our first guess, let's assume that both diodes 1 and 2 are on.

2. **Redraw the circuit with wires for on components and open circuits for off components. Label each wire and open circuit with the diode name.** Now that we've made our assumption about what's on and what's off, we can redraw the circuit appropriately. To keep track of which wire and open circuit correspond to which diode, we'll label each section we redraw.
   In our first step, we assumed that diodes 1 and 2 were on, which means we must redraw them as wires. Let's now redraw the circuit accordingly.



3. **Analyze the redrawn circuit for the currents across the "on" diodes and voltages across the "off" diodes**. Now that we've redrawn the circuit, we solve for the currents passing through the wires that represent "on" diodes and the voltages across the open circuits that represent "off" diodes.
   Here, we must solve for the currents passing through the diode 1 and diode 2 wires. Let's begin our solution by expressing Kirchoff's Laws for different parts of the circuit.



Writing KVL equations around the two loops above, we find:

$$i_{R1}R_1 + i_{R2}R_2 - V_1 = 0 \tag{269}$$

$$V_2 - i_{R2}R_2 = 0 \tag{270}$$

We may also write a KCL equation at the intersection of the two diode wires:

$$i_{R1} - i_{R2} - i_2 = 0 \tag{271}$$

Let's begin the solution process by solving for i_{R2} from the second loop equation.

$$i_{R2} = V_2/R_2 \tag{272}$$
$$i_{R2} = 3\ A \tag{273}$$

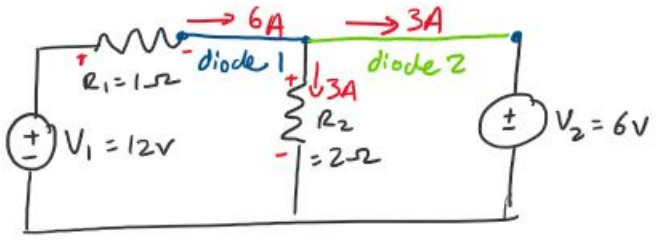
Now, substituting into the equation for the first loop, we may solve for i_{R1}.

$$i_{R1}R_1 + (V_2/R_2)R_2 - V_1 = 0 \tag{274}$$
$$i_{R1}R_1 + V_2 - V_1 = 0 \tag{275}$$
$$i_{R1}R_1 = V_1 - V_2 \tag{276}$$
$$i_{R1} = \frac{V_1 - V_2}{R_1} \tag{277}$$
$$i_{R1} = 6\ A \tag{278}$$

Using i_{R1} and i_{R2}, we may apply our KCL equation and solve for i_2.

$$i_{R1} - i_{R2} - i_2 = 0 \tag{279}$$
$$6 - 3 = i_2 \tag{280}$$
$$i_2 = 3A \tag{281}$$

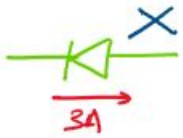
Drawing these currents in on the circuit, we find:



This step is now complete!

4. **Evaluate if the currents and voltages match the on/off assumption. If they do, the problem is complete. If they don't, make a new guess.** In this step, we seek to find if the voltages and currents we just solved for are *consistent* with our diode assumptions. That is: on diodes should have current flowing in the *positive* diode direction (through

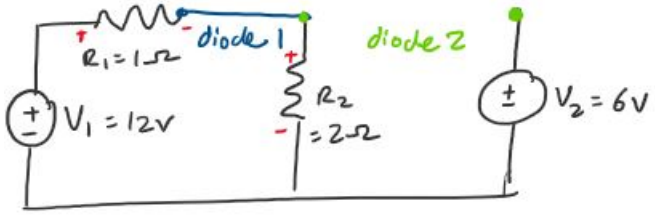
the arrow) and off diodes should have a *negative* voltage across their terminals.
If our circuit analysis *doesn't* match the on/off assumptions, then we know that our initial on/off guesses were incorrect! We then must make a *new* guess for the diode states.
Let's evaluate whether our assumptions were correct for this example. Looking back at the circuit we solved for above, we find that current flows through diode 1 in the positive direction. However, current flows through diode 2 in the *negative* direction - thus, our assumption about the diodes was incorrect!



Above: current can't flow through diode 2 in the reverse direction! Thus, our assumption about the state was incorrect.

Since we know our original guess of on/off was incorrect, let's make a new guess. This time, let's assume that diode 1 is on and diode 2 is off.

5. **Repeat steps 2-4 with the new assumption of diode states**. Now that we have our new assumption (diode 1 on and diode 2 off), we may reanalyze the circuit. First, redraw the circuit:



Now, solve for the current passing through the diode 1 wire and the voltage across the diode 2 open circuit.
Let's start by solving for the current passing through diode 1.

$$i_{R1} = \frac{V_1}{R_1 + R_2} \tag{282}$$

$$i_{R1} = 4\ A \tag{283}$$

Using this current, we may then solve for the voltage across diode 2. We know that one terminal of diode 2 is at 6V, as shown in the redrawn circuit diagram. The voltage of the other terminal is the same as the voltage across resistor 2.

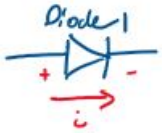
$$V = i_{R2} R_2 \tag{284}$$

$$V = 8\ V \tag{285}$$

Thus, subtracting the voltage from the terminal on one side from the terminal on the other side, we find:

$$V_{diode2} = 6 - 8 = -2\ V \tag{286}$$

Now, let's compare what we've found with our diode state assumptions. First, let's look at diode 1.



As we found in our analysis, diode 1 has a 4A current passing through it in the positive direction. Thus, diode 1 is consistent with its "on" assumption.
Now, let's look at diode 2, which we assumed to be off. Recall that an "off" diode should have negative voltage across its terminals. Looking at our analysis, we find that we have a $-2\ V$ voltage across the terminals. Thus, our second assumption is also consistent!
Because both of our diode assumptions were correct, we know that *this* is the correct state of our circuit. Thus, answering the original question, the following are the currents passing through the diodes:

$$i_{diode1} = 4A \tag{287}$$
$$i_{diode2} = 0A \tag{288}$$

And our solution is complete!

7.2 Transistors

So far in this section, we've looked at a new component, the diode, which is capable of controlling the direction of current. By placing a diode in a circuit, we ensured that current is only able to flow through in a single direction. How else might we want to control the current in our circuit?
Oftentimes in electronics, in addition to controlling the direction of current, we want to control the *amount* of current and *amplify* our signals.
The **transistor** is a specially designed silicon component that's capable of controlling the amount of current in our circuit and amplifying the signal in a very convenient manner.
Because a transistor is capable of *controlling* the amount of current in a circuit,

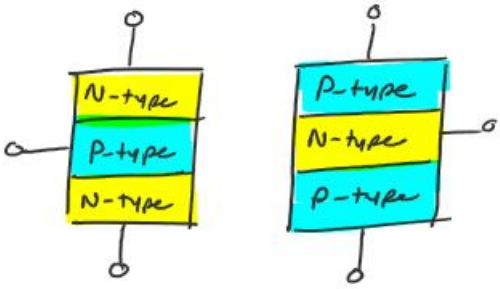
it has the ability to *switch* the current on and off. Because of this, we can use a transistor not only as an amplifier but *also* as an electronic switch.
In this section, we'll introduce two important types of transistors, the Bipolar Junction Transistor (BJT) and the Metal Oxide Semiconductor Field Effect Transistor (MOSFET). We'll also discuss how to analyze circuits with these important components.

7.2.1 Bipolar Junction Transistors

The first type of transistor we'll discuss is called the **Bipolar Junction Transistor**, or **BJT**. This is one of the most common transistors currently in production.

Above: a sketch of a real-world BJT, which you'll find in your lab kit.

A BJT is constructed from layers of "p-type" and "n-type" semiconducting material - special types of silicon with unique electrical properties. These electrical properties are what allow the BJT to function well as an amplifier.
There are two main types of BJT: NPN and PNP. These refer to material type (p or n) of each layer in the transistor. An NPN transistor has a p-type layer sandwiched in between two n-type layers, while a PNP transistor has an n-type layer between two p-type layers.

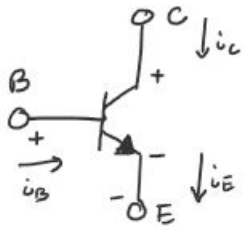


Above: An NPN transistor (left) and a PNP transistor (right)

These two types of transistor have very similar qualities that we'll soon explore further! Let's begin by taking a closer look at the NPN BJT.

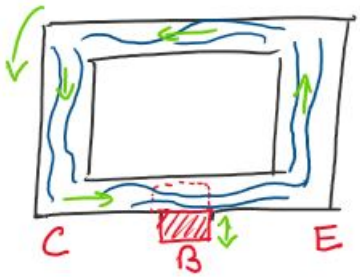
The NPN Transistor

As with all circuit components, we'd like to begin describing the NPN transistor by drawing its circuit diagram symbol, developing its fundamental relationships, and describing its current-voltage relationship.
Let's examine the circuit diagram symbol for the NPN transistor below:



Above: The symbol for an NPN transistor

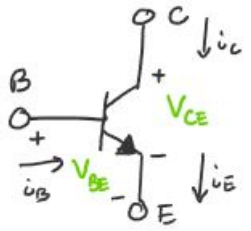
The NPN transistor has three different terminals: **base (B), collector (C), and emitter (E)**.
Let's approach this transistor with a brief fluid flow analogy. Imagine the following stream of water, with a red plug that may be moved up and down to control the rate of flow, is our circuit.



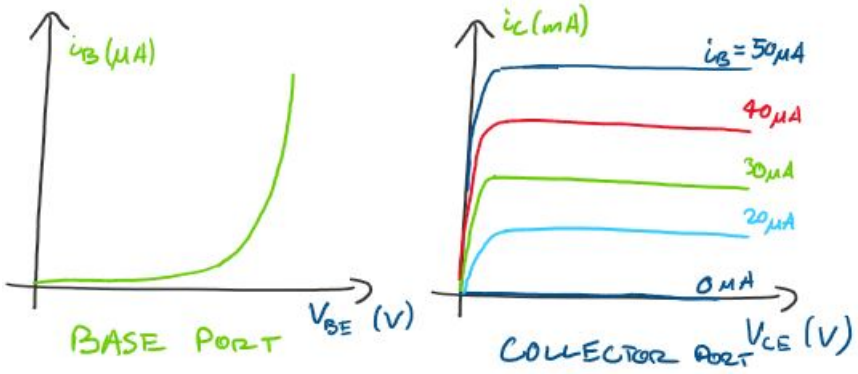
Let's label the movable red plug "B," and the sections on either end of the plug "C" and "E." Let's now zoom in on the section with the plug, and see what happens when we change the plug's position:

We see that when we move the plug (B) higher up, it's much harder for current to flow in the stream. If we were to move it all the way up to the top, the stream would be *blocked* entirely. If it were to be all the way at the bottom, on the other hand, the stream would flow freely.
Thus, we can think of the plug as a *dial* that helps us control our flow. It has the capability to, using a simple input, control how fast our entire stream moves, or whether it moves at all.
An NPN transistor is the *exact* same thing, except for a circuit. By passing in a small input *current* to the base terminal, we can control *how much* current is able to flow between the collector and emitter terminals.
Thus, using the base terminal, we're able to *amplify* our signal or set it entirely

to zero! Thus, we may use our transistor both as an amplifier and as an on/off switch.
Before we describe the current-voltage relationship, let's take a quick second look at the transistor circuit diagram, and establish a convention for what's positive and what's negative for a transistor.



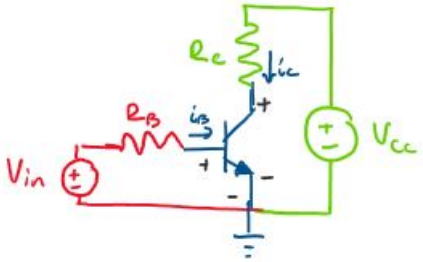
The image above defines the positive, *forward bias* convention for a transistor. The forward bias convention is defined as follows. When the base terminal is at a higher voltage that the emitter terminal, the voltage V_{BE} is positive. When the collector terminal is at a higher voltage that the emitter terminal, the voltage V_{CE} is positive.
The positive current conventions for the NPN transistor are also described in the above image.
Now that we have an understanding of *what* the transistor actually is, let's describe its current-voltage relationship. Transistors are *nonlinear* components, so we describe their I-V relationship with a graph for convenience.[16] Note that for BJTs, there are two current-voltage relationships we'd like to describe: the relationship between V_{BE} and i_B and the relationship between V_{CE} and i_C.



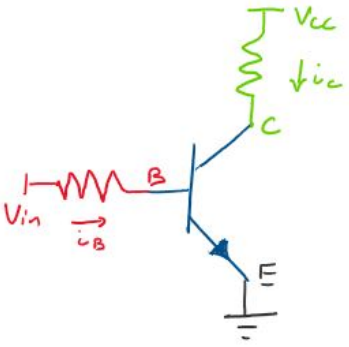
A lot is happening in these current-voltage graphs, and several features, such as the multiple lines, might look unfamiliar! Let's break down what's going on in these graphs.
First, let's look at the graph of V_{BE} and i_B, on the left. We immediately notice

[16] Analytical equations for current and voltage in a transistor *do* exist, but aren't discussed in this course. If you're interested, look into taking a course such as EE 105, Microelectronic Devices and Circuits!

that only a *tiny* current, in units of *microamps*, is passing through the base terminal. This tells us that we only need to pass a tiny amount of current into our base terminal to control the flow of current between C and E.
Let's now look at the second graph, of V_{CE} and i_C. How do we read an I-V relationship with multiple lines? As it turns out, each line represents a different i_B, the controlling input current at the base terminal. Simply select the line that corresponds to the input current at the base terminal and proceed with your analysis using *just* that line.
Make a special note of the units for each variable in the graph on the right. We find that for micro-scale currents being passed into the base port, we get milli-scale currents passing through the collector port. This means that only a very small base current is needed to control the much larger collector-emitter current. This is the power of using a transistor: we can get great control and amplification for only a *very* small input.
How can we actually use a BJT in a circuit? The **common-emitter amplifier** circuit is a typical layout for using a BJT. It appears as follows:

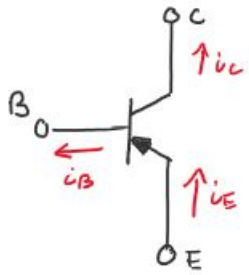


In the common-emitter circuit, we have two loops: the "base" loop and the collector-emitter loop. The base loop has a voltage source, V_{in}, which supplies the base current that controls the transistor. The collector-emitter loop, with current i_c and voltage source V_{cc}, contains the signal to be amplified.
Because this is such a common circuit, it's often redrawn with the following shorthand diagram:



Whenever you see a diagram of this form, know that it's the same as the full circuit diagram above.
How do we actually analyze circuits with BJTs? By using two load-line analyses, the first around the base loop using the base port I-V graph, and the second around the emitter-collector loop using the collector I-V graph, we can find the amplified current and voltage in the output loop of the transistor.
Once we introduce MOSFETs, we'll develop a more detailed solution procedure for transistors and apply it in a practical example! For now, keep this idea of using load line analysis in the back of your mind.

The PNP Transistor
Thus far, we've restricted our discussion of BJTs to the NPN transistor, and haven't talked about the PNP transistor. Let's briefly look at the differences between the two.
Recall that in the NPN transistor, we had two layers of n-type material with a layer of p-type in the center, while in the PNP transistor, we have a layer of n-type material surrounded by two layers of p-type material. What does this mean from a practical standpoint?
Because the transistor is now composed of a different arrangement of materials, many of the definitions for an NPN are simply *reversed* for a PNP transistor. Here, instead of the base terminal having a current *input*, the base terminal *outputs* a current. Additionally, in the circuit diagram symbol, we swap the locations of the emitter and collector terminals, and draw the arrow in the opposite direction to reflect the change.



Above: the circuit diagram symbol for a PNP transistor

7.2.2 Field Effect Transistors

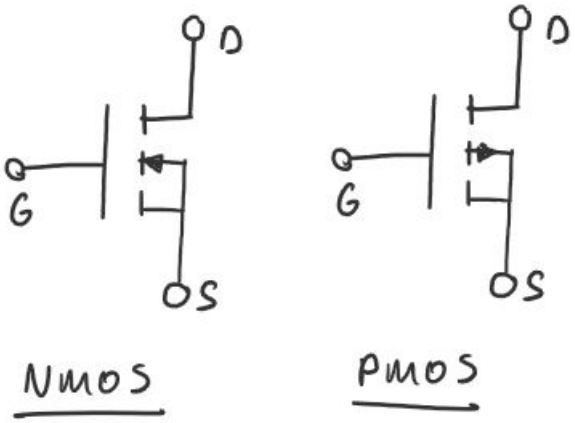
One question that we often face in engineering is: How can we make something more efficient? As in many applications before it, this question was once again asked about the BJT. What followed was the **MOSFET**, the **Metal Oxide Semiconductor Field Effect Transistor**, a more power-efficient, compact, and simple to manufacture transistor.[17]
Because of its compactness and energy efficiency, the MOSFET has made a

[17] Why would you use a BJT given all of the pros of a MOSFET? A BJT is generally better for *high current* applications.

number of advances across electronics possible, and is the main type of transistor used in the field of "digital electronics."

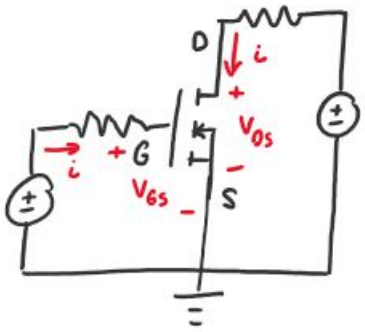
Above: a sketch of a real-world MOSFET, which you may find in your lab kit.

Like the BJT, the MOSFET is still designed around the same basic principle of using specially modified silicon to produce desirable electrical characteristics. After this, however, many of the similarities end! Instead of having the same terminals as a BJT, a MOSFET has 4 different terminals: **drain (D), gate (G), source (S), and body (B)**. It's important to note that a negligible current flows through the body terminal, so in our problems, we often simply connect it to source and treat the MOSFET as a three-terminal device.
We may think of the drain, gate, and source as being analogous to the collector, base, and emitter terminals of a BJT.



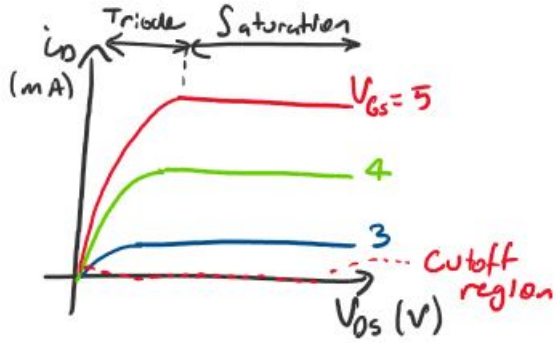
Above: the n-channel (NMOS) and p-channel (PMOS) MOSFETS

As with BJTs, there are two main types of MOSFETs we will see. The first of these is the n-channel MOSFET, or the NMOS. The second type is the p-channel MOSFET, or PMOS. Similarly to BJTs, these names originate from the types (p-type and n-type) and arrangements of silicon used in the fabrication of the transistor.
Similarly to BJTs, MOSFETs are often used with the same simple amplifier circuit, as seen below:



Above: a simple amplifier circuit with an NMOS

Notice that the "forward bias" sign conventions for current and voltage are the *same* for the NMOS as they are for the NPN BJT.
Let's now discuss the current-voltage relationship for an NMOS MOSFET:



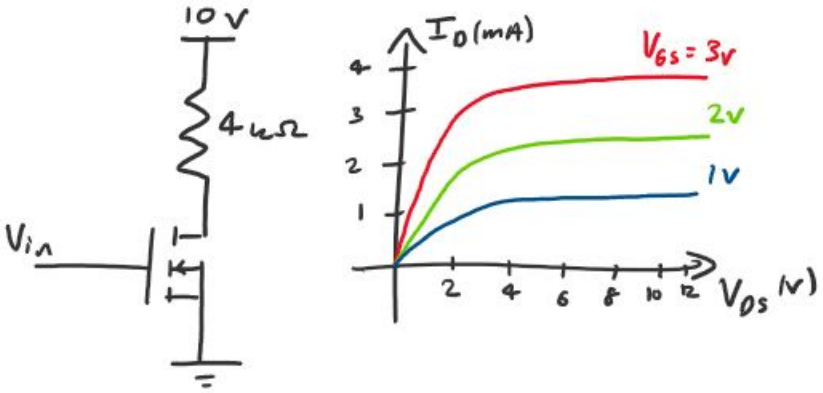
Above: the current-voltage relationship for an NMOS

How do we interpret what's going on in this graph? At first glance, it *appears* to be similar to the NPN transistor graph, but upon closer inspection, we find more nuances to the NMOS transistor I-V relationship.
Each line in the NMOS graph represents a different gate *voltage* (versus current with the NPN). As we can see, the higher the gate voltage is, the higher the output current going through the *drain* will be.
What's most important to note from this graph is what happens when we're on the x-axis, where current is equal to zero. As we can see on the graph, on the x-axis, we have the **cutoff region** for the NMOS transistor. The cutoff region is the region where the gate voltage is not yet high enough to "turn on the switch" and allow current to flow in the source-drain circuit. If the gate is below the **cutoff (threshold) voltage**, V_{to}, the current flowing through the source-drain circuit will *always* be zero.
Looking at the graph, we find that there are several *other* important regions as well! The first of these is the **triode region**. When the MOSFET is in the triode region, the current moving through the gate circuit will *increase* with the

voltage between the drain and source (V_{DS}).
Once V_{DS} becomes too high, however, the MOSFET will enter the **saturation region**, and the current will stop increasing. After the saturation region, the current will remain approximately constant for any input voltage.
In summary: if the voltage is too low, and is below the cutoff voltage, *no current* will flow. If the voltage is above the cutoff region but not too high, current will grow with voltage. If the voltage gets too high, the saturation region will be reached and the current will become constant.

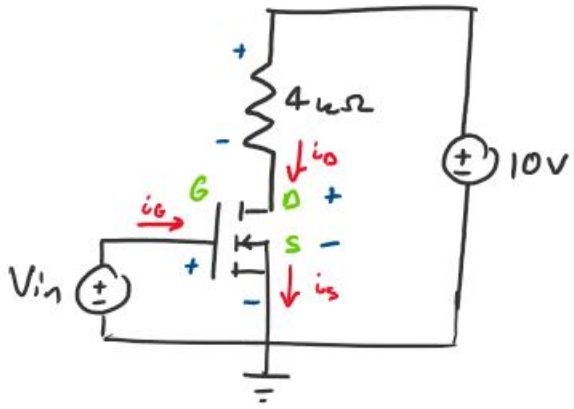
7.2.3 Transistor Analysis

Now that we've defined the details of the MOSFET, we must develop a method of solving circuits with MOSFETs. Let's develop a reliable circuit-solving procedure with the following MOSFET example problem:
Find i_D, the current going through the drain, for the following NMOS transistor circuit. Let $V_{in} = 2\,V$.

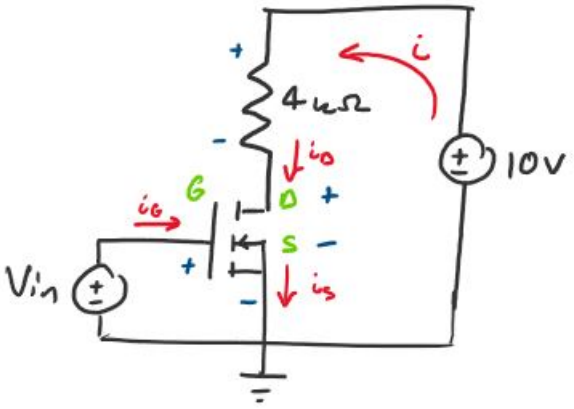


When solving a transistor circuit, we use the following procedure:

1. **Draw out the full circuit diagram**. Recall that for transistors, circuits are often drawn in shorthand like the above. To begin the process of circuit analysis, we must first *redraw* the circuit in a normal circuit diagram.
   We recognize the above circuit as a simple amplifier circuit. Now, we redraw the circuit with voltage sources and complete loops, also taking special care to label the NMOS terminals and their signs.
   First, we replace the 10V node with a 10V voltage source, and connect it to ground. Then, we replace V_{in} with a voltage source and connect it to ground. After labeling the NMOS using our sign convention, our circuit diagram is now complete!



2. **Use KVL to write an equation around the source-drain loop. Solve for current i_D in terms of voltage V_{DS} to find the load line.** Now that we have our circuit diagram in its correct form, we're able to use Kirchoff's Laws to analyze our circuit. By manipulating our circuit equations to get i_D in terms of V_{DS}, we'll be able to get a load line that can plot on our I-V graph.
   Let's begin by writing a KVL equation around the source-drain loop (below):



   Using our KVL sign convention and the signs for voltage we drew in step 1, we arrive at the KVL equation:

$$-10 + i_D R + V_{DS} = 0 \tag{289}$$

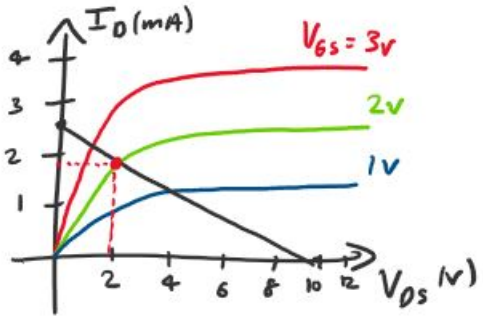
   Plugging in our given values (and leaving current in units of mA), we then

solve for i_D in terms of V_{DS}.

$$-10 + 4i_D + V_{DS} = 0 \tag{290}$$
$$10 - V_{DS} = 4i_D \tag{291}$$
$$2.5 - \frac{V_{DS}}{4} = i_D \tag{292}$$

We now have our load line equation!

3. **Plot the load line on the I-V graph, select the correct V_{GS}, and find the intersection.** Now that we have our load line, all that remains is to plot it on our current-voltage graph and find the intersection. Before we do this, we must select the correct V_{GS} line to find the intersection with.
From the problem statement, we know that $V_{in} = 2\ V$. Looking at our circuit diagram from step 1, we find that this voltage *equals* V_{GS}. Thus, we select the $V_{GS} = 2\ V$ line on the I-V graph and plot our load line:



Finding the intersection between our load line (sketched in black) and our $2\ V$ line, we find:

$$\boxed{i_D \approx 1.9\ mA} \tag{293}$$

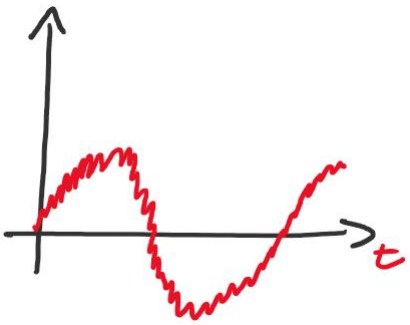
Thus, our problem is complete!

8 Introduction to Digital Electronics

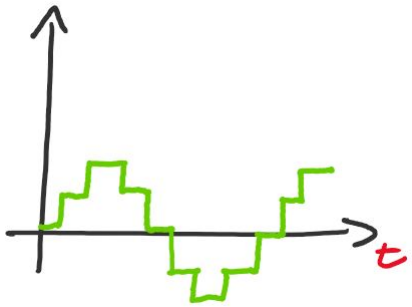
Thus far in our exploration of electronics, we've focused *exclusively* on analog signals: continuous signals that may have any value. In this section of the course, we introduce *digital* signals, signals for which only a select few values are recognized. Here, we'll learn how to analyze this new class of signal with a new class of circuit: the digital circuit.
Along the way, we'll get an introduction to a variety of fields from mathematics and electrical engineering.

8.1 Digital Signals

What *is* a digital signal? To define what exactly a digital signal is and why we would want to use one, let's consider the signal below, which has lots of *random* noise:



Despite the noise, we're able to recognize the familiar profile of a sine wave! How, though, would we be able to *mathematically* describe it as a sine wave? Because of all of the random noise around the sine profile, although it may *appear* to be a sine wave to the human eye, it's *very* challenging to describe it mathematically as such.
This highlights a major problem with practical analog electronics: with just a little bit of random noise, we struggle to accurately describe our signal and lose lots of precision! As we've seen in the past, real-world circuits are complex, non-ideal, and *noisy* entities - how can we deal with signals in a more precise manner?
One answer to this question comes in the form of the **digital signal**. Instead of looking at each individual value as we do for analog signals, we look at **ranges** of values. For example, let's break the sine wave above into a set of discrete ranges. If an analog value is within a certain range, we assign it a constant value. This produces the following digital signal:

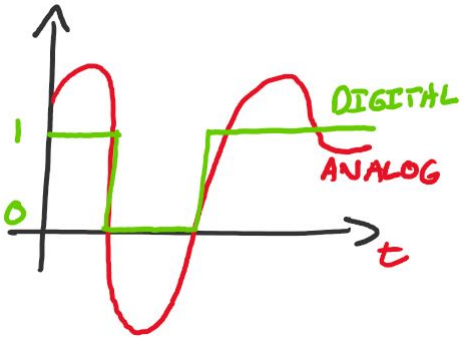


Above: a digital signal with a discrete number of ranges

As we can see, simply by *grouping* our analog values into ranges, we produce a perfectly clean digital signal. This is the power of discrete signals: by defining preset ranges that we care about, versus looking at every single value, we can gain much more precision in our signals.

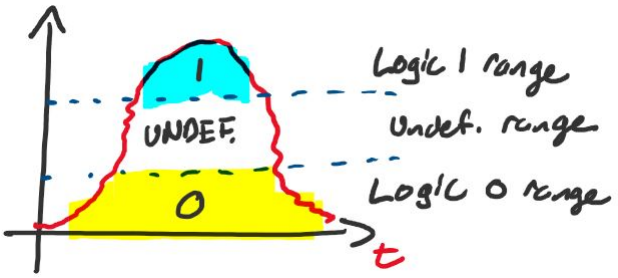
8.1.1 Binary Signals

One of the most convenient, and by far the most common, type of digital signal is the **binary signal**. A binary signal is a digital signal with only *two* ranges. Whereas the digital signal in the graph above had many different ranges defined, a binary signal only has two, which are assigned **logic values** of 0 and 1.



Above: a binary signal, which has values of either 0 or 1

In the example above, we define the 1 range to be positive values, and the 0 range to be negative values. Looking at the original analog signal, we may sketch out our binary digital signal.
Typically, when working with binary signals, the higher amplitude is assigned to the binary "logic" value 1. A logic value 1 is also called **high**, **true**, or **on**. The lower amplitude in a binary signal is typically assigned to the logic value 0, which is also known as **low**, **false**, or **off**. As we proceed, keep these alternate definitions in the back of your mind - we'll come back to them shortly!
How do we define what the ranges for logic 1 and logic 0 are?



For any analog signal, we'll typically define the cutoffs for logic 1 and logic 0 in a way similar to the image above. Logic 0 starts at a low value (in this case 0), and extends up to the 1st cutoff. Logic 1 starts at a 2nd cutoff, at a greater than but not equal to value to the first, and continues upward. Between the logic 0 and logic 1 cutoffs, we have the *undefined* region. Anything we get in this region is considered "meaningless" and is *not* assigned a logic value.

8.1.2 Binary Numbers

Now that we've defined a binary signal, let's see how we can actually *represent* one. Each individual binary digit (1 or 0) is known as a **bit**. By combining bits, we make **digital words**, for example:

$$101101010$$

The 1st bit in a digital word has a special name: the **most significant bit (MSB)**. Since we call the 1st bit the *most* significant bit, we naturally refer to the last bit in a digital word the **least significant bit (LSB)**.
Since the bit is such a small unit, we define the **byte**, which is equivalent to 8 bits. When discussing the length of digital words, we typically use bits or bytes.
Once we begin representing digital signals using binary words, however, we notice something interesting! A binary word is *actually* a binary number! This fact will enable us to unlock the true power of digital electronics.
Let's review what a binary number is. You may recall from your math class that we use a **base 10** number system. This means that we can represent any number as a power of 10 multiplied by a constant less than 10 and greater than or equal to 0. For example, we can represent the numbers 13 and 326 in base 10:

$$1 \cdot 10^1 + 3 \cdot 10^0 = 13 \tag{294}$$

$$3 \cdot 10^2 + 2 \cdot 10^1 + 6 \cdot 10^0 = 326 \tag{295}$$

By writing the coefficient of each power of 10 side by side, we form a number in base 10. Looking at the coefficients, we see that the first number is 13 and the second number is 326 in base 10.
Binary numbers are numbers in a **base 2** number system. Instead of powers of 10, we use powers of 2. Our coefficients, which must be less than 2 and greater

than or equal to 0, will therefore be either 0 or 1.
Let's try representing the number 13 in binary. Breaking 13 up into powers of 2, we see:

$$1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 13 \tag{296}$$

Looking at the coefficients of the powers of 2 and writing them side by side, we find that 13 represented in base 2 is 1101.
Let's come up with a reliable algorithm to convert a number from decimal to binary with the following example:
Convert the decimal number 26 into binary

1. **Floor divide the number by 2 and find the remainder.** Our first step will be to "floor divide" our decimal number by 2 and record the remainder. Floor division involves dividing two numbers and rounding *down* to the nearest integer. In each step, the remainder will be one of our binary digits. In this case:

$$floor(26/2) = 13 \tag{297}$$
$$remainder = 0 \tag{298}$$

2. **Floor divide the result by 2 and record the remainder.** After our first division, our number is now 13. We now floor divide this by 2 again and record the remainder.

$$floor(13/2) = 6 \tag{299}$$
$$remainder = 1 \tag{300}$$

   Here, we have a remainder of 1, since 13 does not evenly divide by 2.

3. **Repeat steps 1 and 2 until the number equals 0**. We continue to floor divide the number and record the remainder each time until the number equals 0.
   First, floor divide 6 by 2 and record the remainder:

$$floor(6/2) = 3 \tag{301}$$
$$remainder = 0 \tag{302}$$

   Now, floor divide 3 by 2 and record the remainder:

$$floor(3/2) = 1 \tag{303}$$
$$remainder = 1 \tag{304}$$

   Finally, floor divide 1 by 2 and record the remainder:

$$floor(1/2) = 0 \tag{305}$$
$$remainder = 1 \tag{306}$$

   Now that we've reached 0, this step is complete!

4. **Write the remainders from last to first**. Starting with the last remainder and ending with the first, write out the remainders. What results will be our binary number!
   In this case, we start with the first remainder, which came from 26/2, and continue until the last to write:

$$\boxed{11010} \tag{307}$$

   This is 26 in binary, and is our final answer!

How do we convert back into decimal? Recall how we defined binary numbers: as coefficients multiplied by powers of 2. Earlier, for example, we represented the number 13 as:

$$1 \cdot 10^1 + 3 \cdot 10^0 = 13$$

Whenever we want to go from our binary representation *back* into decimal, we simply write out our binary number as a sum of coefficients multiplied by powers of 2 and find the result!
For our binary representation of 26, for example, we would write:

$$1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0$$

Peforming this calculation, we get 26, our decimal number.

8.1.3 Alternative Number Systems

Something you may have noticed is that although binary is very simple, as they can represent any number with just 0s and 1s, they get *very* large very quickly! Because of this, binary is not the only number system used in electronics! Two other number systems, termed "octal" (base 8) and "hexadecimal" (base 16) are also commonly used!
Let's examine hexadecimal a little closer - as you'll soon see, hexadecimal has some interesting quirks compared to other number systems. In the table below, you can find decimal numbers from 0 to 15 and their Hexadecimal equivalents:

Decimal	*Hexadecimal*
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9

Decimal	*Hexadecimal*
10	A
11	B
12	C
13	D
14	E
15	F

As you can see, after 9, we stop representing numbers by *digits* and begin using *letters* instead! Why is this? This is because in any base number system, we want to be able to represent each number up to our base with a single character. In base 10, for example, we were able to write the numbers 0 through 9 using single digits, and in base 2, we wrote 0 and 1 with single digits.
In base 16, we want to do the same thing, but have the unique challenge that there *aren't* enough single digit numbers! To work around this, we use letters instead.

8.2 Boolean Algebra

Thus far, we've discussed binary signals through the lens of *binary numbers*. How else may we look at a binary signal? Is there any other way we can take advantage of their unique structure?
Let's think back to our definition of binary signals. Recall that a binary signal is defined as having two ranges: 0 and 1. You may remember that we *also* provided alternate names for 0 and 1: **false** and **true**.
Instead of thinking of binary signals as just numbers, we instead interpret 1 to mean "true" and 0 to mean "false," and discover a whole new perspective on binary signals. This interpretation leads us to the subject of **Boolean Algebra**, which concerns the manipulation of true and false values.
When working with "normal" algebra, we often assigned names, called variables, to numbers to make our work clearer. In Boolean algebra, we do the same! A **logic variable** is written as a capital letter by convention, and can have a value of either 1 (true) or 0 (false). For example, we can define the logic variables:

$$A = 0 \tag{308}$$
$$B = 1 \tag{309}$$
$$C = 1 \tag{310}$$

Just as in regular algebra, where we negated, added, and multiplied our variables, we may do the *same* for our logic variables! Let's define these three operations for logic variables.
The negation operator, known as the **complement**, is written with an overbar and is defined as follows:

$$\overline{1} = 0 \tag{311}$$
$$\overline{0} = 1 \tag{312}$$

As you can see, the complement *flips* the value of a logical variable. The complement operator is also known as "**not**." If you see the expression $\overline{A}$, for example, where A is a logical variable, you can read it out as "not A."
Let's now discuss logical addition. If we have two logic variables, A and B, we may logically add them by writing $A + B$. Logical addition is also referred to as the **or** operation. As such, when you see $A + B$ written out, you may read it as "A or B."
Logical addition is defined by the following rules:

$$0 + 0 = 0 \tag{313}$$
$$1 + 0 = 1 \tag{314}$$
$$0 + 1 = 1 \tag{315}$$
$$1 + 1 = 1 \tag{316}$$

Just as we defined negation and addition, we may also define logical multiplication, also known as the **and** operation. When we have two logical variables A and B, we may represent their product by writing $A \cdot B$, or AB for short. Because of the alternate name for multiplication, we can also read AB as "A and B."
We define the different cases for multiplication as follows:

$$0 \cdot 0 = 0 \tag{317}$$
$$1 \cdot 0 = 0 \tag{318}$$
$$0 \cdot 1 = 0 \tag{319}$$
$$1 \cdot 1 = 1 \tag{320}$$

As you can see, even with the simple operations of addition and multiplication of two logic variables, there are *lots* of different cases to keep track of! How can we stay organized when computing the value of logic expressions?
One method to do this is to use a logical **truth table**. A truth table is a table describing *all* possible outcomes of a logical expression.
Let's construct a truth table for each of the above operations! Let's begin with the complement operation.
Create a truth table for $\overline{A}$, where A is a logical variable.
When creating a truth table for a logic expression, we may use the following procedure:

1. **Make a column for each term in the logical expression.** Our first step is to lay out the structure of our table. We can do this by drawing a table with a column for each term in the logical expression.
   Your table should be organized as follows: write all of the basic logical variables (such as A, B, C) that appear in your expression *all the way* to the left of your table and write any other expression to the right. Separate your logical variables and the more complex expressions with a vertical line to stay organized.

Here, we write our basic logic variable, A, in the leftmost column of a table. We write the logical expression, $\overline{A}$, to its right.

A	$\overline{A}$

2. **Write out all combinations of the logic variables**. After creating our basic table, we must write out every possible combination of our logic input variables. Here, we only have 1 variable, so we only have two possible values.

A	$\overline{A}$
0	
1	

3. **Moving from left to right, evaluate all the remaining logical expressions.** Now that we've filled in all our logical variables, we evaluate all the remaining expressions to complete our truth table. At the end, we'll have a complete description of our expression!

A	$\overline{A}$
0	1
1	0

Since they tell us what our expression will be for *every* possible combination of inputs, truth tables provide us with a complete understanding of our expression. Let's now apply the truth table procedure to a more interesting example: the addition operator.
Make a truth table for $A + B$, where A and B are logical variables.
Let's follow the procedure we just defined:

1. **Make a column for each term in the logical expression.** Here, we have two basic logic variables, A and B. We write A and B in columns in the left of our table. Then, drawing a vertical line, we make a column for the more complex compound expression, $A + B$, on the right side of the table.

A	B	$A+B$

2. **Write out all combinations of the logic variables**. Now that we have our columns for the table, we must write out all of the possible combinations of A and B. Note that since we have two variables, our logic table will be *larger* than that for the complement. Note that in general, if

you have an expression with n logic variables, your truth table will have 2^n rows.

A	B	$A+B$
0	0	
1	0	
0	1	
1	1	

3. **Moving from left to right, evaluate all the remaining logical expressions.** Now that we've filled in all the possible combinations of A and B, we're ready to write out the truth table for our remaining expression, $A+B$.

A	B	$A+B$
0	0	0
1	0	1
0	1	1
1	1	1

Our truth table for the + operator is now complete! As you can see, $A+B=1$ when *either* A or B or *both* A and B are 1.

The same process may be applied to produce the truth table for the logical multiplication operator:

A	B	AB
0	0	0
1	0	0
0	1	0
1	1	1

Why use a truth table? Truth tables are especially valuable when we have a complex expression that we want to find all possible values of. For example, if we have the following expression:

$$A(B+C)+B \tag{321}$$

We can easily find its values by writing out the truth table! When writing the truth table for complex expressions, it can often be useful to write out the values of the smaller sub-expressions as well instead of jumping straight to the last step. This helps us stay organized with the different parts of our expressions. For the expression above, for example, we may find it useful to make columns for the sub expressions $B+C$ and $A(B+C)$ so we can easily keep track of them! Using this idea of creating columns for sub-expressions, we could write the truth table for $A(B+C)+B$ as:

A	B	C	$B+C$	$A(B+C)$	$A(B+C)+B$
1	1	1	1	1	1
1	1	0	1	1	1
1	0	1	1	1	1
1	0	0	0	0	0
0	1	1	1	0	1
0	1	0	1	0	1
0	0	1	1	0	0
0	0	0	0	0	0

Notice that by making columns for smaller sub-expressions, we were able to find the values of larger expressions with much greater ease!
Also note that since we have 3 logical variables, we have $2^3 = 8$ rows in our truth table! This follows our rule of having 2^n rows for n logical variables.

8.2.1 Proving Statements with Truth Tables

When we learned basic algebra in our math courses, after learning the fundamental rules of adding, multiplying, and negating variables, we learned a number of useful algebraic properties that helped us *simplify* expressions.
The distributive property, for example, allowed us to turn the expression $x(y+z)$ into the equivalent expression $xy + xz$ by *distributing* the variable x.
When we develop more and more advanced expressions in Boolean algebra, it becomes *very* useful to develop a similar set of properties. How, though, can we prove that two expressions are equivalent using Boolean algebra?
By showing that two expressions have the **same truth table** for every combination of input variables, we can decide if two expressions are equivalent. If they have the same truth table, they're equivalent, if not, they're non-equivalent!
Let's try this out with the following example:
Prove the distributive property holds in Boolean algebra by showing $A(B + C)$ is equivalent to $AB + AC$.
Let's now write out the truth tables for $A(B + C)$ and $AB + AC$. If we get it right, we'll see that the two have equivalent truth tables, and we'll know that they're equivalent Boolean expressions.
Instead of writing out two entirely separate truth tables, we can write the two expressions in the same truth table and simply check if their columns are equal.
Let's start by setting up the different columns for our truth table! Let's use some extra columns to keep track of the values of some of the subexpressions.
Filling in the truth table using our previously established methodology, we find:

A	B	C	$B+C$	AB	AC	$AB+AC$	$A(B+C)$
1	1	1	1	1	1	1	1
1	1	0	1	1	0	1	1
1	0	1	1	0	1	1	1
1	0	0	0	0	0	0	0
0	1	1	1	0	0	0	0
0	1	0	1	0	0	0	0
0	0	1	1	0	0	0	0
0	0	0	0	0	0	0	0

As we can see, the columns for $AB + AC$ and $A(B + C)$ are identical for every combination of A, B, and C. Thus, $AB + AC$ and $A(B + C)$ are equivalent expressions, and the distributive property holds for Boolean algebra.
The distributive property isn't the only useful fact we can prove! In the table below, you'll find a list of useful laws in Boolean algebra. For x and y logical variables:

Expression	*Name*
$\overline{\overline{x}} = x$	*Law of the double complement*
$x + \overline{x} = 1$	*Unit property*
$x\overline{x} = 0$	*Zero property*
$x(y + z) = xy + xz$	*Distributive law*
$x + x = x$ $x \cdot x = x$	*Idempotent laws*
$x + 0 = x$ $x \cdot 1 = x$	*Identity laws*
$x + 1 = 1$ $x \cdot 0 = 0$	*Domination laws*
$x + y = y + x$ $xy = yx$	*Commutative laws*
$x + (y + z) = (x + y) + z$ $x(yz) = (xy)z$	*Associative laws*
$\overline{xy} = \overline{x} + \overline{y}$ $\overline{x + y} = \overline{x}\,\overline{y}$	*De Morgan's laws*
$x + xy = x$ $x(x + y) = x$	*Absorption laws*

These laws can all be proved using truth tables! Make a particular note of **De Morgan's Laws**, which tell us that the addition and multiplication operations are actually related by the complement operator!

8.2.2 Proving Statements with Laws

Thus far, we've been using truth tables to work with logical variables and prove statements of equivalence. What happens, however, when the number of variables begins to grow? For example, if we have five logical variables, by our 2^n

rule, we would have $2^5 = 32$ rows in our truth table!
Since this would be very impractical to create, when working with more complex logical expressions, we often *stay away* from truth tables and seek to prove statements about logical expressions purely by relying on the laws of Boolean algebra.
By applying the laws we just defined in clever ways, we can *manipulate* Boolean expressions into equivalent ones! This idea of manipulating Boolean expressions will soon come into use when we work with complex systems of logic gates.
When proving equivalence using the expressions of Boolean algebra, note that there's rarely one single way to complete a problem - thus, unlike circuit analysis, there's no specific procedure that will get us the answer every time.
Instead of a specific procedure, we define a general set of steps that can help guide us in our Boolean algebra proofs:

1. **Expand**: Look for patterns in your expression that resemble one of the Boolean algebra laws. Apply the law to expand the expression.
2. **Eliminate**: After expanding, see if you can eliminate any terms in your expression.
3. **Simplify**: Reduce your remaining expression into the simplest form possible using Boolean algebra laws.

By repeating these 3 steps until we get our desired expression, we can reliably write Boolean algebra proofs. Let's now do an example of proving equivalence using the laws of Boolean algebra.
For p and q logical variables, show that $\overline{(p + (\overline{p}q))}$ and $\overline{p} \cdot \overline{q}$ are equivalent logical expressions.
When solving these problems, it's generally good practice to begin with the more complex expression. Let's begin with the first: $\overline{(p + (\overline{p}q))}$. Looking at the expression, we notice that we can apply De Morgan's Laws to the two terms inside the parentheses. Let's try this and see where it takes us:

$$\overline{(p + (\overline{p}q))} = \overline{p} \cdot \overline{(\overline{p}q)} \tag{322}$$

Now we focus on the new expression, $\overline{p} \cdot \overline{(\overline{p}q)}$. How can we reduce this further? Let's apply De Morgan's Laws once again, this time to the expression in parentheses on the right!

$$\overline{p} \cdot \overline{(\overline{p}q)} = \overline{p} \cdot (p + \overline{q}) \tag{323}$$

Applying the distributive law to the result, we see:

$$\overline{p} \cdot (p + \overline{q}) = \overline{p} \cdot p + \overline{p} \cdot \overline{q} \tag{324}$$

Recognizing that $\overline{p} \cdot p = 0$, we write our final expression:

$$\overline{p} \cdot p + \overline{p} \cdot \overline{q} = \overline{p} \cdot \overline{q} \tag{325}$$

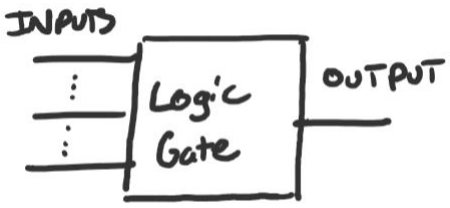
Thus, through a series of equivalent expressions, we've shown that:

$$\overline{(p + (\overline{p}q))} = \overline{p} \cdot \overline{q} \tag{326}$$

And without any truth tables, our problem is complete!
Notice how looking for recognizable clusters of terms and seeing if we could *expand* them was a strong strategy! Once we began expanding our expression, by continuing to recognize patterns in the results, we were able to complete the proof.[18]

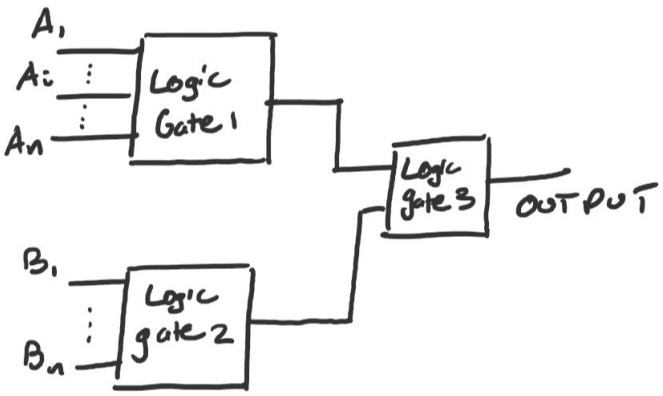
8.3 Logic Circuits

So far, we've discussed binary signals and Boolean algebra in an almost purely mathematical capacity. Let's now bring the theory that we've developed back into the world of electronics, and discuss how we can apply Boolean algebra to create digital logic circuits.
Logic circuits are a way to represent Boolean algebra operations and expressions in circuit-diagram form. Instead of writing an expression out using variables, we may draw it using circuit symbols that represent real-world components. This approach helps us translate the abstract mathematical approach of Boolean algebra into something much more grounded in reality.
The core component of a logic circuit is the **logic gate**, a component that performs logical operations on a signal. Generally, a logic gate will appear as follows:



On the left-hand side, we have any number of inputs to the logic gate, which are logic variables. The logic variables enter the logic gate, which performs some logic operation on them. On the right-hand side, the result of the operation is put out.
By connecting different logic gates together, we form **logic circuits** that represent complex Boolean operations.

[18] Interested in learning more about Boolean algebra or proof strategies for other tough problems? Courses like Math 55 (Discrete Mathematics) and CS 70 (Discrete Mathematics & Probability Theory) go in-depth on both.

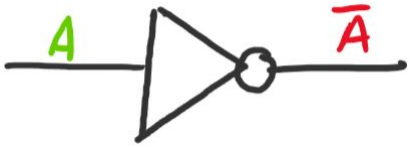


Above: an example of a logic circuit with 3 gates. Notice how the outputs of the first two logic gates feed into the input of the third logic gate.

Just as we had many different circuit components in analog electronics, there are many different types of logic gates that can go into a logic circuit. Let's now discuss some of the different logic gates.

8.3.1 The Logic Inverter

Since we began our study of Boolean algebra with the complement operator, we will begin our study of logic gates with the **logic inverter** logic gate. The logic inverter is a logic gate that performs the complement operation on its input. Mathematically speaking, if a logic variable A is passed into the input of the logic inverter, the output will be $\overline{A}$.
The symbol for a logic inverter, a triangle with a circle on the end, is the following:



Above: The logic inverter. Notice that the logic inverter only has a single input and single output.

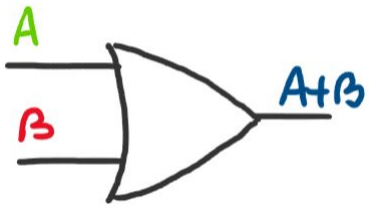
Because all logic gates represent logical operations, we may write out truth tables for logic gates. The truth table for the logic inverter is the same as the truth table for the complement. For A the logic variable input to the gate, the truth table for the inverter is thus as follows:

A	$\overline{A}$
0	1
1	0

Just as we defined a logic gate corresponding to the complement, we can also define logic gates corresponding to logical addition and multiplication!

8.3.2 The Or Gate

The logic gate that performs logical addition is the **or gate**. Recall that when defining logical addition, we mentioned that $A + B$ can be read as "A or B." It's for this reason that we call the logic gate for addition the "or" gate! The symbol for a two-input "or" gate is found below:

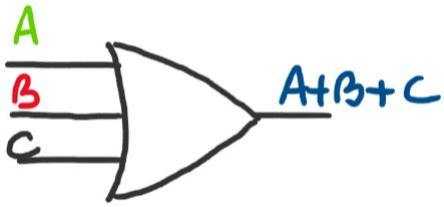


Above: a 2-input or gate. Two logical variables enter on the left side, and their logical sum is output on the right side.

The truth table for an or gate is thus the same as the truth table for addition:

A	B	$A + B$
0	0	0
1	0	1
0	1	1
1	1	1

Just as with addition, an or gate will *only* return 1 when *either* A or B or *both* A and B equal 1.
Although the "or" gate in the above example only had two inputs, an "or" gate can have *any* number of inputs! For example, if we wanted to represent the expression $A + B + C$, we could use an "or" gate with 3 inputs:

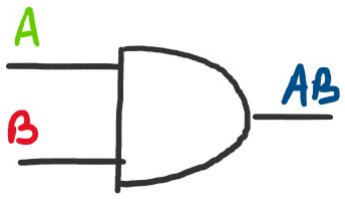


Above: An example of an or gate with 3 inputs. Note that no matter what, an or gate can only have a single output.

How would we calculate $A + B + C$ with 3 inputs? We follow the same rule as always for the addition operator. If at least one logic variable is equal to 1, the expression evaluates to 1.
The or gate may be compared to the **or** command in Python, which we use in conditional if-then statements! Recall that when one of the expressions in an or statement is true, the entire expression is true. This same concept is used by an or gate - if one input variable equals 1, the or gate outputs 1.

8.3.3 The And Gate

Let's move on to the next gate: the **and gate**. The "and" gate represents Boolean multiplication. If we have logic variables A and B passed into the "and" gate, their product, AB, will be returned.
The symbol for the and gate is as follows:



Above: A two-input and gate.

Just as with the or gate, an and gate may have any number of inputs. Recall that for a Boolean multiplication operation to return 1, all of the inputs must be 1. Because of this, you can think of the and gate as you would the **and** operator in Python, where all of the sub-expressions must be true for the full expression to be true.

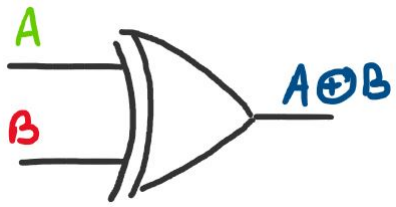
8.3.4 Other Logic Gates

Although the inverter, "or", and "and" gates are the most common logic gates, there are others that can still prove to be useful! All of the following logic gates can actually be constructed from the logic gates we've discussed thus far, and simply provide us with "shortcuts" to other commonly used logic operations.[19]
The **XOR**, or "exclusive or" gate is the first of these that we'll discuss. The XOR gate is based on the following idea: imagine that you tell your friend "I'm going to the movies *or* going for a run."
When you tell your friend this, he knows that you're going to do either one of these activities, *not* both! However, thinking about our definition of the or gate, this sentence could mean that we're going to the movies *and* going for a run. This is because for logic variables A and B, "A or B" is equal to 1 when either A or B is equal to 1 or when *both* A and B are equal to 1.
The XOR gate provides us with a way to express the "exclusive or" - the XOR gate will return 1 only when *either* A or B is 1, and will return false if *both* A and B are 1. This logic gate expresses what we're trying to communicate to our friend when we say "or."[20]
The XOR operation has a special symbol: $\oplus$. If we have two logical variables A and B, and we want to express "A XOR B," we would write $A \oplus B$. The truth table for $A \oplus B$ is as follows:

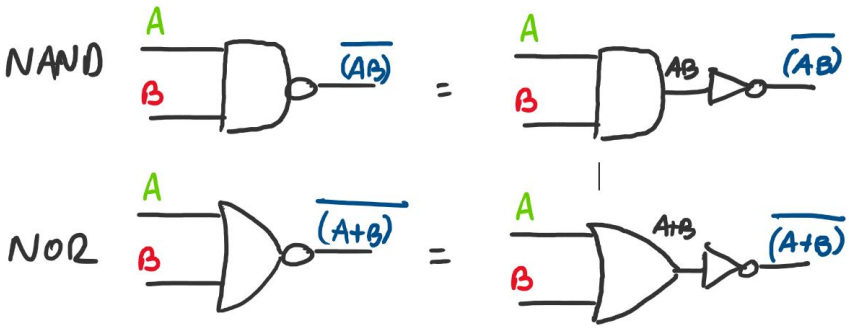
[19] De Morgan's Laws tell us that we only actually need the inverter and an "or" or "and" gate to construct all other gates - "or" and "and" are related by the inverter.
[20] You should really tell your friend "I'm going to the movies XOR going for a run."

A	B	$A \oplus B$
0	0	0
1	0	1
0	1	1
1	1	0

We may draw the symbol for the XOR logic gate as follows:



Notice the similarity in symbols between the XOR and or gates.
Let's discuss two more gates: the **NOR** gate and the **NAND** gate. In logic circuits, the negation operation is exceedingly common - as such, it's convenient to define gates that negate the outputs of the "or" and "and" gates. This is all that the NOR and NAND gates are - "or" and "and" gates followed by an inverter!
The two gates, as well as their equivalent representations with inverters, are depicted as follows:

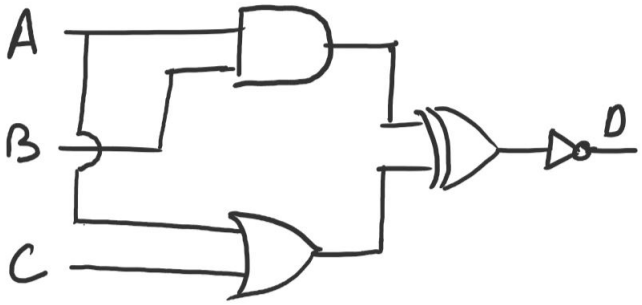


Because they're simply "or" and "and" gates followed by an inverter, the truth tables for NOR and NAND are simply the same as the "or" and "and" truth tables, only with each value flipped.
This gives the following truth table, where columns have been made for "or" and NOR as well as "and" and NAND.

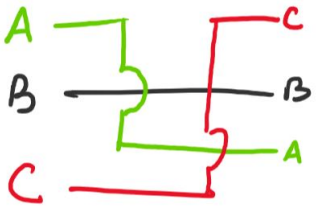
A	B	$A+B$	$\overline{(A+B)}$	AB	$\overline{(AB)}$
0	0	0	1	0	1
1	0	1	0	0	1
0	1	1	0	0	1
1	1	1	0	1	0

8.3.5 Logic Circuit Analysis

By connecting large numbers of logic gates together, we can build logic circuits that perform a number of useful tasks in electronics, for example adding numbers in binary. These logic circuits form the foundation of computer architecture, and are the building blocks for the computers we use everyday.
Because of their wide range of applications, it's important to understand how logic circuits perform. In this section, we take a first look at the analysis of logic circuits, a first step in our eventual goal of designing our own logic circuits.
Before we jump into logic circuit analysis, let's briefly discuss some conventions for drawing logic circuits. A logic circuit will typically appear in a form similar to the following:



We have all of our different input variables on the far left, our gates in the middle, and an output variable all the way on the far right.
To show which logic variables go into which gate, we use "wires" that start at the symbol on the far left and connect to the desired gate.
Note that due to the layout of logic circuits, these "wires" will often cross over each other. To avoid confusion when it comes to this, we draw small "jumps" in the wire whenever two wires would cross over each other.



Above: We draw wires as "jumping" over each other wherever they would otherwise cross.

Oftentimes in logic circuit analysis, we're faced with the following problem: how can we go from a Boolean expression to a logic circuit diagram? In the following simple example, we develop a methodology for translating Boolean expressions into logic circuit diagrams.
Draw the logic circuit diagram for the Boolean expression: $D = AB + \overline{C}$*, where A, B, C, and D are logic variables.*

When drawing the logic circuit diagram from a Boolean expression, we may use the following procedure:

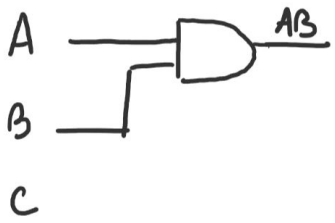
1. **Identify all of the input and output logic variables.** Before we begin drawing a circuit, it's important to identify all of the variables we're going to be working with. The output variable is the logical variable all on its own on one side of the expression, and is what we want our circuit to evaluate to.
   The input variable(s) are all of the variables on the other side of the expression. They're what we're *manipulating* using logic gates to reach the output.
   In this case, we identify our output variable as D and our input variables as A, B, and C.

2. **Draw the input variables on the left side of your diagram.** We can begin our logic circuit sketch by writing out all of our input variables in a column in alphabetical order.
   For our example, this would produce the following start to our sketch:

3. **Identify all sub-expressions.** Now that we've identified all of our logic variables and started our sketch, we can begin forming the framework for our logic gates.
   We start by identifying all of the sub-expressions of our Boolean expression - the smaller expressions within our larger Boolean expression. This helps us *break down* a complex Boolean expression into simpler parts. Note that it's not uncommon for sub-expressions to contain *further* sub-expressions within themselves.
   Note that there's not always a single right answer when choosing sub-expressions! There are lots of ways to look at Boolean expressions, especially in more complex ones.
   Here, where $D = AB + \overline{C}$, we identify two sub-expressions: AB and $\overline{C}$.

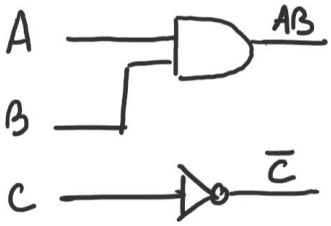
4. **Implement the sub-expressions using logic gates.** Now that we've identified the sub-expressions of our Boolean expression, we can return to our circuit diagram.
   Going one sub-expression at a time, we represent each sub-expression using logic gates. Let's begin with the first sub-expression, AB. Using our sketch from step 2, we can draw the logic gate that produces the simple

expression AB.
Recognizing that AB is a product, we use an *and gate.* Starting from the positions for A and B we sketched out, we draw "wires" from A and B going into an and gate.

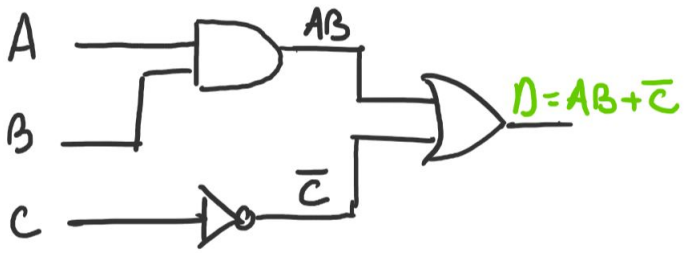


Above: the logic gate for the first sub-expression. Remember to leave enough space for the other logic gates in your drawing!

After labeling the output of the and gate as AB, we move onto our second sub-expression, $\overline{C}$. We recognize that this sub-expression can be represented by an inverter gate. Drawing this on our logic circuit diagram and labeling the output, we get:



5. **Connect the sub-expressions.** To form the full Boolean expression, all that remains is to correctly connect the sub-expression circuits.
Recall that our expression, $D = AB + \overline{C}$, joins the two sub-expressions, AB and $\overline{C}$, with a logical addition operation. Since we represent addition with an "or" gate, we connect the outputs of our two sub-expressions with an or gate. Completing this and labeling the output, we arrive at our final logic circuit diagram:



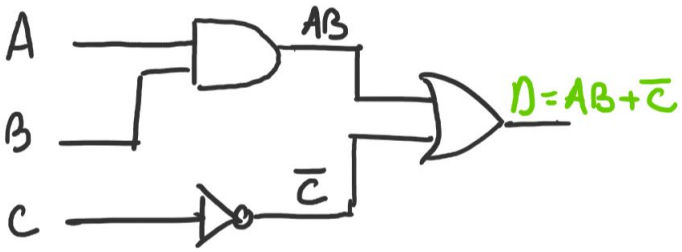
We now have a complete representation of our Boolean expression!

9 Digital Circuit Design

Previously, we began to explore the fascinating world of digital electronics. Taking a sharp left turn from the phasors and systems of equations that ruled analog circuitry, we looked at Boolean algebra, and began to learn about the different types of logic gates. We also learned how to put these logic gates together into circuits, and translate from *abstract* mathematical expressions into logic gates that can be implemented in the real world.
In this chapter, we'll dive deeper into digital electronics, and ask the fundamental question of how we can *design* different digital circuits to fit our needs. Additionally, we'll learn about a new digital circuit component: the flip flop.

9.1 Designing Logic Circuits

Previously, we began discussing how to construct *digital logic circuits* from logic gates. By combining logic gates in certain ways, we were able to construct circuits that performed Boolean operations.
For example, recall that we were able to represent the expression $D = AB + \overline{C}$ by the following logic circuit:



So far, in logic circuits such as the above, we've been provided with a Boolean logic expression such as $D = AB + \overline{C}$, and have been asked to come up with the corresponding logic circuit.
What if we're asked to complete a more *challenging* task? What if instead of being provided with a Boolean expression, we're provided with a *truth table*, and have to come up with a logic circuit that replicates the truth table's behavior?

A	B	C	D
1	1	1	1
1	1	0	1
1	0	1	1
1	0	0	0
0	1	1	0
0	1	0	0
0	0	1	0
0	0	0	0

For example, if we're presented with the truth table above, how would we design a logic circuit that produces the output D? As it happens, this is a common

problem in digital electronics. Oftentimes, we don't know exactly *what* Boolean expression we want, but rather have a set of desired input and output combinations.
By using the laws of Boolean algebra, we can develop clever strategies to enable us to design logic circuits for *any* truth table, no matter how complex!
When converting a truth table into a logic circuit diagram, we'll use the following general two-step procedure.

1. **Determine the Boolean expression from the truth table.** Using the laws of Boolean algebra, develop a Boolean expression that produces all of the desired outputs in the truth table.

2. **Convert the Boolean expression into a logic circuit.** Using our previously developed strategies for sketching logic circuits, convert the Boolean expression from step 1 into a logic circuit.

9.1.1 Finding Boolean Expressions

Let's now discuss some different strategies for completing step 1: how can we find a Boolean expression from *just* a truth table? We'll answer this question through the following example:
Find a Boolean expression and logic circuit for the truth table:

A	B	C	D
1	1	1	1
1	1	0	1
1	0	1	1
1	0	0	0
0	1	1	0
0	1	0	0
0	0	1	0
0	0	0	0

When looking at a truth table like this and deciding a viable method to find the Boolean expression for D, the output, we might resort to something like "guess and check." By guessing Boolean expressions combining A, B, and C that look like they could produce the output D, we *might* be able to find our answer. But, for this problem, this involves guessing eight rows correctly! This is a tall order that only becomes more challenging as we add variables.
Instead of resorting to guess and check to determine an expression for D, we may use the following procedure, known as the **sum of products** method.

1. **Identify all of the rows where $\mathbf{D = 1}$.** Our first step is to identify all of the rows in our truth table where D equals 1. By doing this, we take one step closer to making sure D will have the right value in all possible cases.
   Looking at the truth table above, we identify that the first, second, and third rows all have $D = 1$.

2. **For each row identified in step 1, write a product of the inputs that makes D = 1.** Now that we've identified all of the rows where $D = 1$, we want to figure out *how* to get $D = 1$ for each case by *multiplying* the input variables.
Why do we use multiplication? Because logical multiplication is only equal to 1 for *one* combination of input variables, we know that we won't accidentally produce other combinations that give a value of 1.
Let's begin with the first row where $D = 1$:

A	B	C	D
1	1	1	1

Our goal is to write some product of A, B, and C that produces D. Here, A, B, and C are all equal to 1. Thus, we may write:

$$D = A \cdot B \cdot C \tag{327}$$

This is our first logic expression. Let's now do the same for the other rows we identified in step 1.
Next up is row 2:

A	B	C	D
1	1	0	1

Here, although $C = 0$, D is *still* equal to 1. How can we write a product of A, B, and C to make $D = 1$? We simply *invert* the value of C. Our expression for row 2 is thus:

$$D = A \cdot B \cdot \overline{C} \tag{328}$$

For row 3, our final row from step 1, we have the following:

A	B	C	D
1	0	1	1

Here, since $B = 0$, to get $D = 1$, we invert B. This produces the following expression:

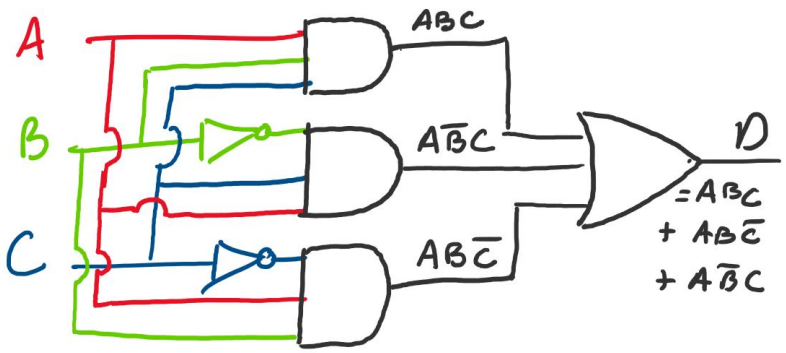
$$D = A \cdot \overline{B} \cdot C \tag{329}$$

3. **Sum all of the expressions for D from step 2.** Once we have all of our Boolean expressions that represent *all* of the ways we can combine A, B, and C to make $D = 1$, we simply add them up! This gives us our final Boolean expression:

$$\boxed{D = ABC + AB\overline{C} + A\overline{B}C} \tag{330}$$

Because this expression will have a value of 1 for *every* correct combination of A, B, and C and will have a value of 0 otherwise, it fully represents our truth table. We've now found our mystery Boolean expression!

4. **Draw the logic circuit diagram.** Using our logic circuit drawing techniques, we may sketch a logic circuit for the Boolean expression we just found. For this problem, our circuit will be:



   We've now successfully designed a logic circuit for a truth table! Our problem is now complete.

Let's take a moment to step back and think about *why* this worked. In step 3 of our procedure, why did we sum up all of our expressions for $D = 1$? How can we be sure that none of the other combinations in the truth table would equal 1?

Let's think for a moment about the truth table for the following Boolean expression: $F = X + Y + Z$, for X, Y, Z, and F logical variables.

X	Y	Z	F
1	1	1	1
1	1	0	1
1	0	1	1
1	0	0	1
0	1	1	1
0	1	0	1
0	0	1	1
0	0	0	0

As we can see in the truth table, whenever one or more of X, Y, or Z equal 1, the *entire* expression is true. When none of the X, Y, and Z equal 1, the expression is false.

In the sum of products method, we take advantage of this useful structure to find our Boolean expression. We know that when one of the sub-expressions, which corresponds to a row in our truth table, equals 1, the entire expression will equal 1. Similarly, when none of them equal 1, the entire expression will equal 0.Since we only get 1 for the 3 cases we described, every other combination of variables will lead to an output of 0! Therefore, by adding together all of the cases where D should equal 1, we *must* get a logical expression that satisfies all

of the cases.
As it happens, the sum of products method is *not* the only method for going from a truth table to a Boolean expression and circuit diagram. Another method, the **product of sums** method, focuses on the rows of the truth table that are equal to *zero*.
Let's discuss the product of sums method for the same example:
Find a Boolean expression and logic circuit for the truth table:

A	B	C	D
1	1	1	1
1	1	0	1
1	0	1	1
1	0	0	0
0	1	1	0
0	1	0	0
0	0	1	0
0	0	0	0

Product of sums is defined by the following procedure:

1. **Identify all rows where D = 0.** Instead of focusing on the rows where $D = 1$, as we did with the sum of products method, for product of sums, we turn our attention to the rows where $D = 0$.
   For this example, rows 4, 5, 6, 7, and 8 all have $D = 0$.

2. **For each row identified in step 1, write a logical sum of all the variables that equals 0.** Whereas in sum of products, we found *products* of the input variables for all of the $D = 1$ rows, for product of sums, we find a sum of all of the input variables for the $D = 0$ rows.
   Remember, for a sum of logic variables to equal 0, *all* of the logic variables must equal 0. Let's begin with row 4, the first row we identified:

A	B	C	D
1	0	0	0

   To get $D = 0$ by adding the input variables, we would write:

   $$D = \overline{A} + B + C \tag{331}$$

   We now repeat this process for all of the remaining zero rows:

   $$D = A + \overline{B} + \overline{C} \tag{332}$$
   $$D = A + \overline{B} + C \tag{333}$$
   $$D = A + B + \overline{C} \tag{334}$$
   $$D = A + B + C \tag{335}$$

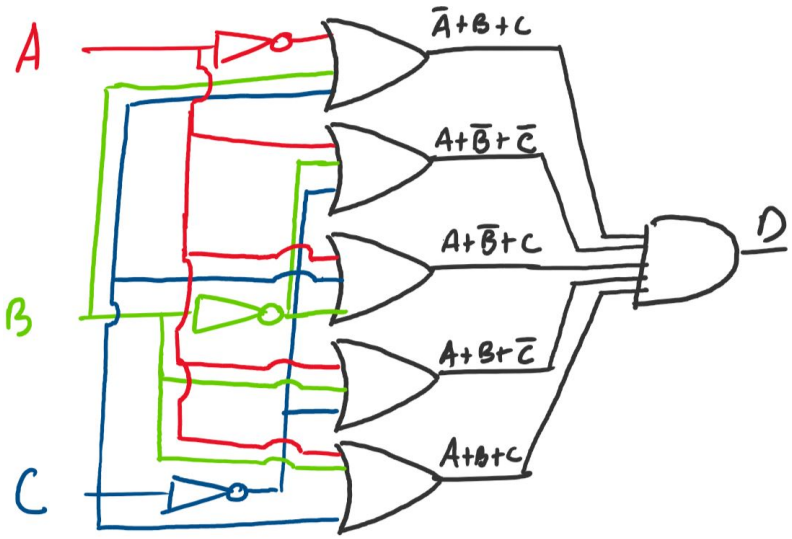
   This step is now complete!

3. **Multiply all of the Boolean expressions from step 2.** By taking the product of all of the Boolean expressions from step 2, we find our final Boolean expression:

$$\boxed{D = (\overline{A} + B + C)(A + \overline{B} + \overline{C})(A + \overline{B} + C)(A + B + \overline{C})(A + B + C)} \tag{336}$$

   You might notice that this expression for D is *different* to what we found using the sum of products method - this is actually expected! For every truth table, there are always multiple ways to write a Boolean expression for D.

4. **Draw the logic circuit diagram.** Using our logic circuit diagram drawing techniques, we arrive at the following diagram:

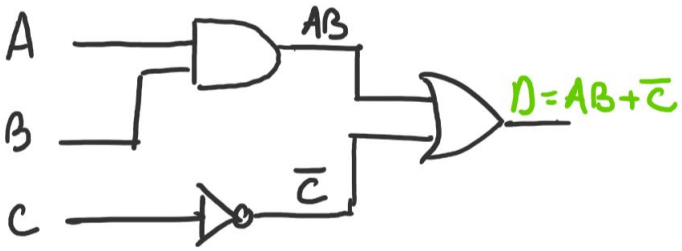


   And our problem is complete!

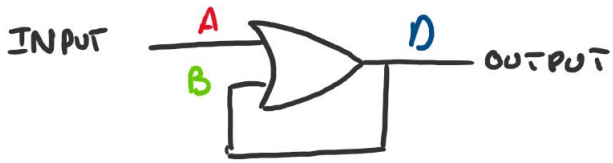
You may have noticed that for this truth table, our product of sums circuit diagram, which we just drew, was significantly more complex than our sum of products diagram despite representing the same truth table. How can we tell which method will give us the *simplest* circuit diagram?
When the output column of a truth table (D in the example above) has fewer 1s than 0s, it's a good idea to use sum of products. Since sum of products focuses on the rows equal to 1, this will result in fewer logic gates. On the other hand, when the output column fewer 0s than 1s, it's good practice to use product of sums, as it focuses on the zero rows. Although this is a good guideline to follow, note that you may *always* apply both methods to any truth table.

9.2 Sequential Logic Circuits

Thus far, our logic circuits have all been of the following form: a series of input variables that pass into a set of logic gates, which then operate on the input variables to form some output.
Across all of our logic circuits, one thing has remained constant: our circuits have *no memory* of their past inputs. For the following circuit, for example:



If at one point in time, $A = 0$, $B = 1$, and $C = 0$, and at another point in time, $A = 0$, $B = 1$, and $C = 1$, the output of the circuit at time 2 will *not* be impacted by the output at time 1 - every input is totally independent.
Because of this, we say that Boolean logic circuits are **memoryless**: they have no recollection of their past states. **Sequential logic circuits** are a class of circuit that *do* have memory: their values will depend on the past states of the circuit.
How can we design these sequential logic circuits? To begin, we'll construct the foundations of a component known as the **flip flop**, a part that forms the basis for many sequential logic circuits.
First, consider the following digital circuit:



This circuit, which uses an or gate, has a distinctly different setup to what we've seen before. Instead of having two separate inputs and a single output, we have a single input and an output that *connects back* to one of the input terminals. Let's analyze the behavior of this circuit for a couple of different cases.
Imagine that the circuit begins in the following state, where A, B, and D all equal 0.

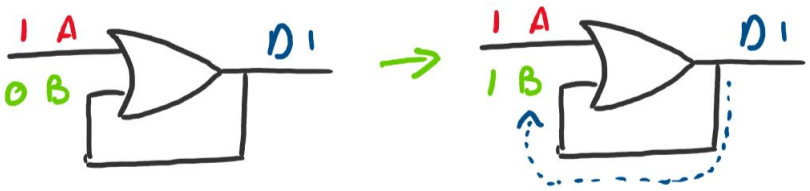
A	B	D
0	0	0

In this case, A and B are equal to 0, which makes D also equal to 0. This is consistent with the definition of the or gate.
Something interesting, however, happens when we change A to 1. From the

definition of an or gate, we know that having $A = 1$ leads to an output of $D = 1$.

A	B	D
0	0	0
1	0	1

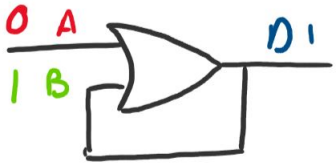
However, because the output of D *feeds back* into B, the state of B must change to 1.



Thus, right after this state, we get the next row of the truth table, where B is now equal to 1:

A	B	D
0	0	0
1	0	1
1	1	1

Let's ask another interesting question. What will happen if we change A back to 0? Since B and D are *already* equal to 1, their value will *not* change, leaving us with a circuit in the following state:



Thus, if we change A back to 0, we will get the final row:

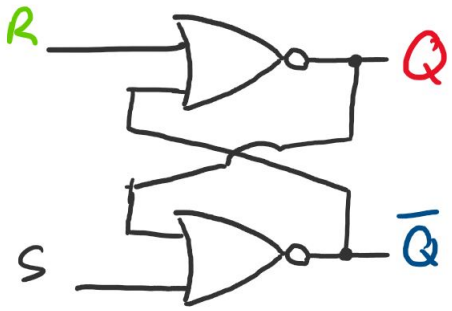
A	B	D
0	0	0
1	0	1
1	1	1
0	1	1

In this case, since they remained connected to each other, B and D *remembered* their value from the previous state, and stayed the same! The value of the *new* state depended on the previous state, and we thus say that the circuit thus has **memory**.

When working with sequential logic circuits, we construct more complex versions of these circuits that "connect back" to each other using different logic gates. These circuits are known as **flip flops**, and come in many different forms.

9.2.1 The SR Flip Flop

The first type of flip flop we'll discuss is the **SR (Set/Reset) flip flop**. The logic circuit diagram for the SR flip flop is found below:

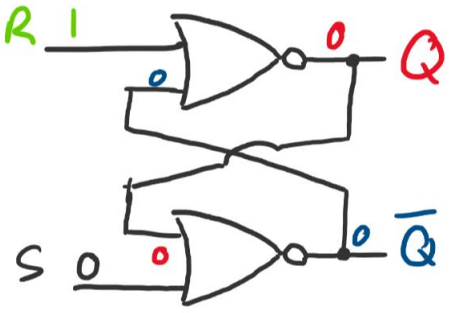


Above: An SR flip flop, constructed from two NOR gates

In an SR flip flop, we have two inputs, S and R, on the left-hand side, and two outputs, Q and $\overline{Q}$, on the right-hand side. We'll soon see that Q and $\overline{Q}$ are always complements of each other.
Let's characterize the behavior of the SR flip flop by looking at different combinations of R and S and observing the resulting outputs at Q and $\overline{Q}$. As an SR flip flop is constructed from two NOR gates, we provide the NOR gate truth table below as a reminder to help in our analysis:

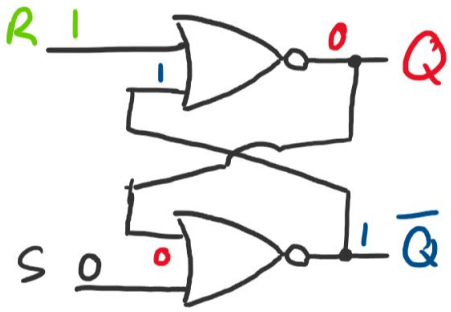
A	B	$\overline{(A+B)}$
0	0	1
1	0	0
1	1	0
0	1	0

Let's begin by testing out the following condition: assume all logic variables except R are equal to 0.



Let's evaluate where this circuit "settles." If R is 1 and the other terminal of the NOR gate is 0, looking at the NOR gate truth table, Q will stay at 0. But,

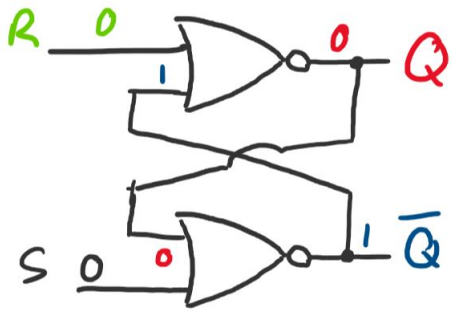
looking at the bottom NOR gate, if both inputs are 0, then $\overline{Q}$ becomes 1. This puts the circuit in the following state:



Let's check to see if this is the final state of the circuit for this combination of R and S. If we have both inputs to the top NOR gate as 1, the output of the NOR gate, Q, will equal 0. Thus, the circuit remains in the current configuration. Now that we have the final configuration for this combination of R and S, we fill in the first row of our SR flip flop truth table:

R	S	Q
1	0	0

Now, let's evaluate what happens to Q when we turn both R and S off. Starting from our current configuration, let's change R to 0.



Above: notice how Q and $\overline{Q}$ always have opposite states

As we can see in the diagram above, when we change R to 0 and keep S at 0, *nothing* happens - the circuit *remembers* its last state and stays there. We now write this into the truth table with a special syntax:

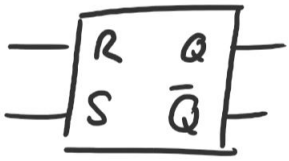
R	S	Q_n
1	0	0
0	0	Q_{n-1}

Instead of just writing Q, we use Q_n to denote the present state of Q. For the case we just discussed, where R and S are 0, we use Q_{n-1} to express that Q

keeps its previous state. With flip flops, it's common to use subscripts like these to express the state a part of the circuit is in.
Analyzing the circuit for the remaining combinations of R and S, we get the final truth table for our SR flip flop:

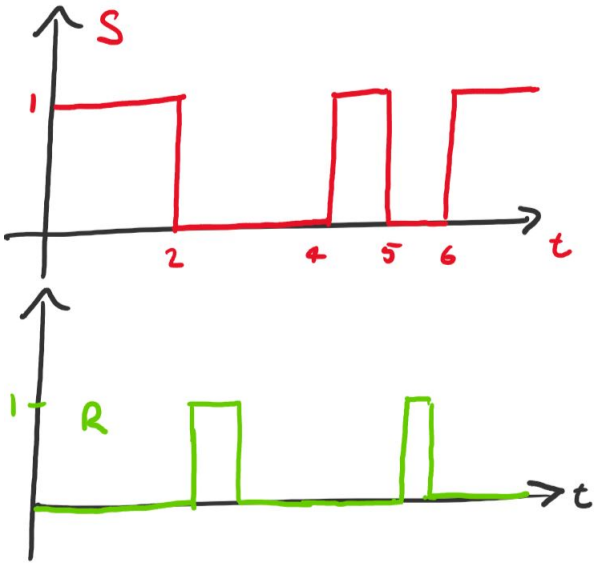
R	S	Q_n
1	0	0
0	0	Q_{n-1}
0	1	1
1	1	*Not Allowed*

Note that having both R and S equal to 1 would make Q and $\overline{Q}$ to be at the same value. Since this state would contradict the two being complements of each other, we say that inputs of $R = S = 1$ are "not allowed."
When we want to draw an SR flip flop in a logic circuit, since the actual NOR gate circuit can be cumbersome to draw, we use the following shorthand symbol:

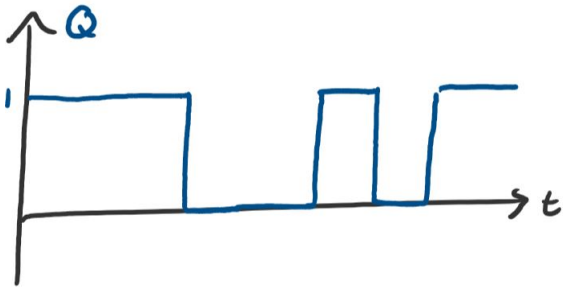


Above: the SR flip flop circuit symbol is a square with four terminals

Notice how in the circuit symbol, all that we do is *replace* the NOR gate circuit with a square box that represents the circuit inside.
Let's now do a quick example of an SR flip flop problem:
Given the following waveforms for S and R as functions of time, plot Q as a function of time for an SR flip flop.



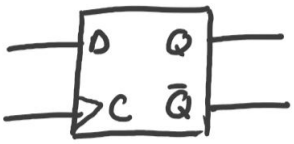
Here, our task is to plot the graph of Q corresponding to the provided graphs for S and R. All we need to do to plot Q is apply the rules of the truth table! According to the truth table, when S is 1 and R is 0, Q will be 1. When S is 0 and R is 1, Q will be 0. Further, when both S and R are 0, Q will stay the same as before.
With these rules in mind, by comparing the values of S and R at each point in time, we plot the following graph for Q:



9.2.2 The D Flip Flop

Let's discuss a second, equally important type of flip flop: the **D (delay) flip flop**. In our discussion of the SR flip flop, and our example problem, we notice that our output signal Q, was *entirely* dependent on the inputs, S and R. What if in addition to having input signals, we also had a way to control *when* the flip flop changes its state?
A **clock signal** is a signal composed of regular pulses that control when our flip flop will change its state. An example of a clock signal is shown below:

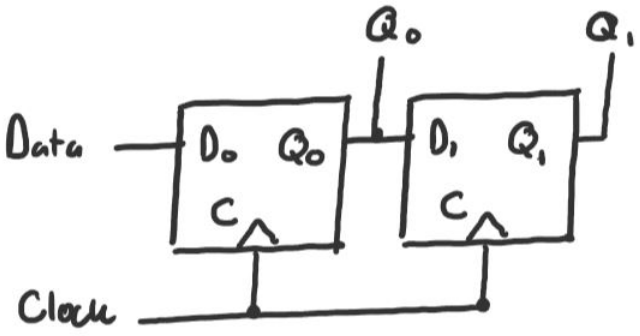
A clock signal has two important "edges" in its waveform. When the signal goes from *low to high*, we say that it's at a **leading edge**, or a "**positive-going edge**." When the signal goes from *high to low*, we say that it's at a **trailing edge**, or a "**negative-going edge**."
A D flip flop is known as a **positive edge triggered** flip flop. This means that it updates its state at *every* positive-going edge. When drawing a D flip flop in a circuit, we use the following symbol:

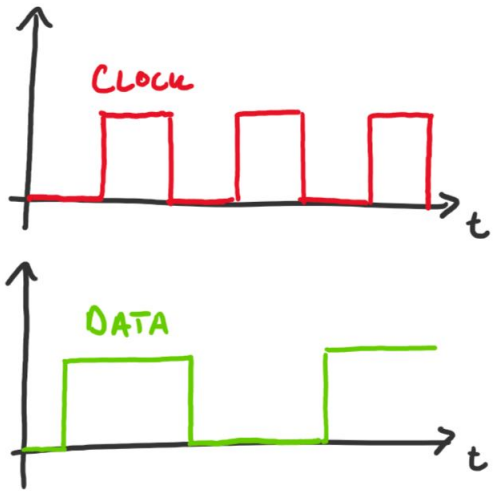


On the left hand side, we have the data signal input, D, and the clock input, C. The small triangle on the clock input indicates that the D flip flop is an edge triggered flip flop. On the right hand side, we have Q and $\overline{Q}$ as outputs of the flip flop. Note that we'll often omit the $\overline{Q}$ port, as we don't always need it in our circuit.
We describe the D flip flop's behavior with the following truth table:

C	D	Q_n
0	$\times$	Q_{n-1}
1	$\times$	Q_{n-1}
$\uparrow$	0	0
$\uparrow$	1	1

Let's break down what this truth table is telling us line by line. The first line tells us that if the clock signal, C, is constant at 0, no matter what the data signal is, the state will always be the previous state. Similarly, from the second row, if C is constant at 1, no matter what the data signal is, the state will remain at the previous state.
From the 3rd row, we find that when we're at a positive-going edge of C, indicated by $\uparrow$ in the truth table, and the value of D is 0, the state will update to be 0. Similarly, from row 4, when we're at a positive-going edge of C and the value of D is 1, the state will be equal to 1.
Let's observe the D flip flop in action with the following example:
Given the clock and data signals for the circuit below, sketch the waveforms for Q_0 and Q_1. Assume that Q_0 and Q_1 start at 0.

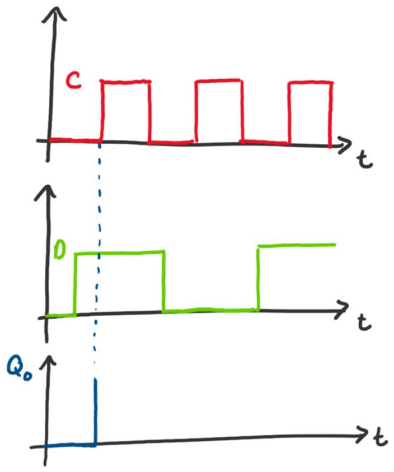




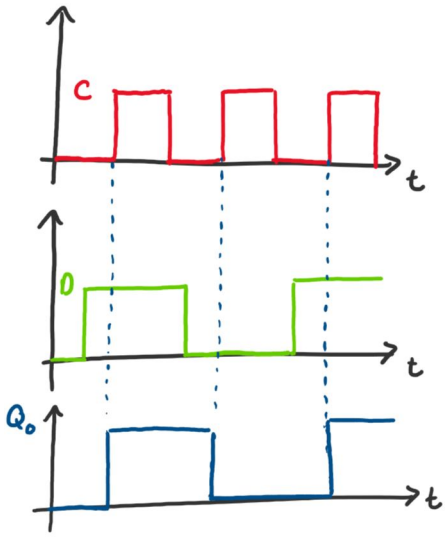
Note that this circuit is actually a very common layout in digital electronics known as a **serial-in parallel-out shift register**. By passing in a single data input, we get out multiple parallel data outputs, each with waveforms shifted from each other. We'll soon see how the D flip flop enables this behavior.

Let's begin by sketching a waveform for the 1st output, Q_0. We know for a D flip flop that at *every* leading edge, the value of Q_0 will be updated to equal the value of D at that time. From the prompt, we *also* know that Q_0 starts at 0. Thus, we may begin our sketch by drawing a line starting at 0 and continuing until the *first* leading edge of the clock. Once we're at the leading edge, we check the value of the data signal, and find that it's equal to 1. Thus, we *update* the value of Q_0 to be 1.

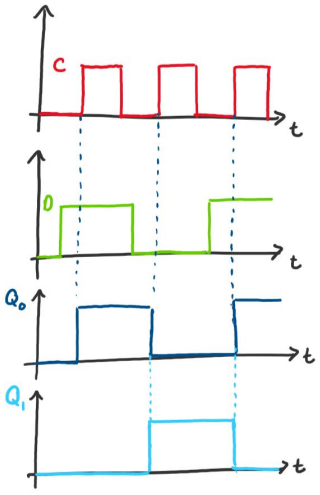
When completing this process, it's helpful to draw a dotted vertical line starting at the leading edge and going down through D up to Q_0. This helps us find where Q_0 will change as well as what its new value will be.



Now, Q_0 continues to be at 1 until the next leading edge, where we check the value of the data signal again and update Q_0 accordingly. We repeat this until we reach the end of the provided signals. This gives us the following:



Thus, we have a complete waveform for Q_0! Now, we want to find the output for Q_1. Looking back at our circuit, we find that the data input for Q_1 is actually the *output* of Q_0. Thus, all that remains is to repeat the process using Q_0 instead of D.
Completing this, we arrive at our final answer:



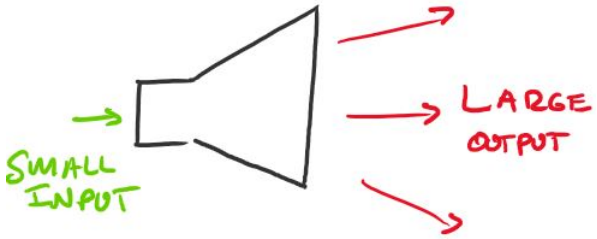
As you can see, at each step, the data wave is *shifted* over! This is what gives the serial-in parallel-out *shift* register its name. Our problem is now complete!

10 Operational Amplifiers

In the past, we've performed extensive analyses of AC, DC, and digital signals, as well as their use in circuits. In this section, we explore methods of manipulating these signals and *amplifying* their strength.
We'll begin with a discussion of the amplifier, the class of component that makes this amplification possible, and then progress into an analysis of an important type of amplifier: the op amp.

10.1 Amplifiers

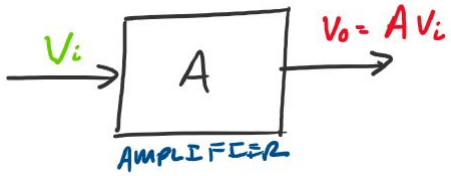
Imagine you're faced with the following circuit design challenge. You're using a microcontroller to power a loudspeaker, but due to the limited output of the microcontroller, your speaker is far quieter than you wish it to be.
How can you design a circuit that *amplifies* your microcontroller signal to produce a louder yet equivalent tone? To accomplish this task and design a successful circuit, we would use an electrical **amplifier**.
When using an amplifier in a circuit, our goal is the following: by inputting a small current or voltage into an amplifier, we want to get out a greatly amplified signal with the same waveform.



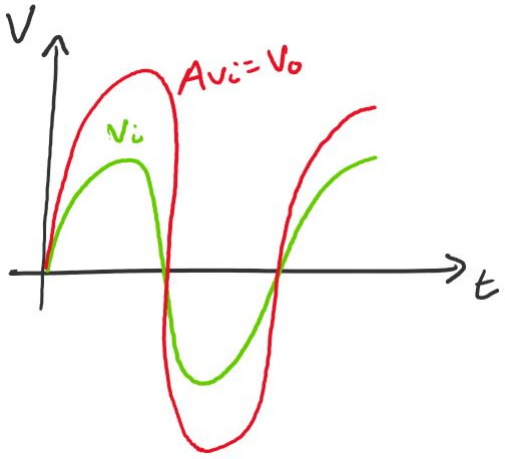
Above: by entering a small input signal into a loudspeaker, we'd like to get a larger version of the signal out.

10.1.1 The Simple Amplifier

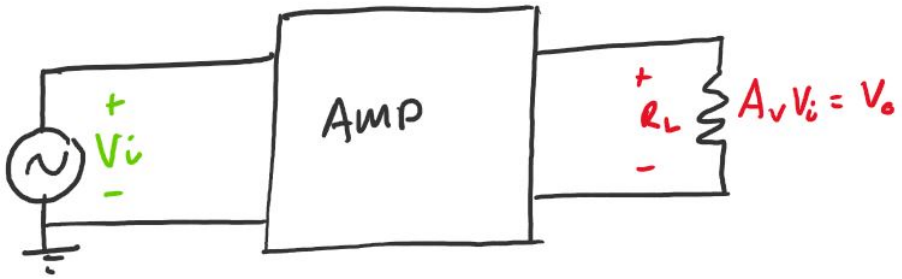
Let's begin developing the background for this special component with a study of the **simple amplifier**, an amplifier with a single input, a single output, and a constant amplification.
At the most fundamental level, we can think of amplifiers in terms of *block diagrams*. Consider the following block diagram for the simple amplifier:



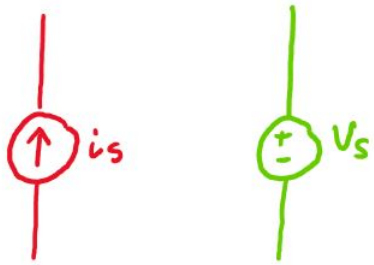
Let's break down what's happening in this diagram, and observe how we may use it to understand how a simple amplifier functions.
On the left of the block diagram, we have the input signal, V_i, represented by an arrow. This input signal is then scaled by some constant A as is passes through the amplifier. The output signal, $V_o = AV_i$, exits the amplifier on the other side.
To describe *how much* amplification occurs, we define the **amplifier gain**. The gain of an amplifier is the ratio of the amplitude of the output signal to the amplitude of the input signal, and is the scaling factor the input is multiplied by to get the output. For the simple amplifier, for example, the gain would be the scaling factor A.
In practice, how would this amplification look for a waveform? Consider the following graph of a simple sine wave, where the input signal is in green and the amplified output signal is in red.



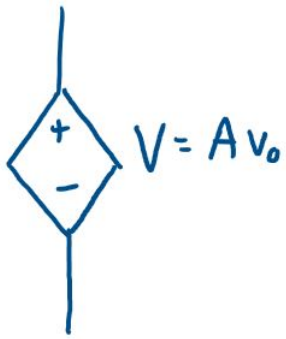
As you can see in the graph above, when a simple sine wave is passed into an amplifier, the *same* wave, simply with a greater amplitude, comes out on the other side. All other properties, such as frequency and phase, remain unaffected.
Now that we have this simple picture of a block diagram in mind, let's take one step closer toward an actual circuit. Instead of using a simple block diagram approach with arrows for input and output signals, let's think about amplifiers in terms of voltage inputs and outputs.



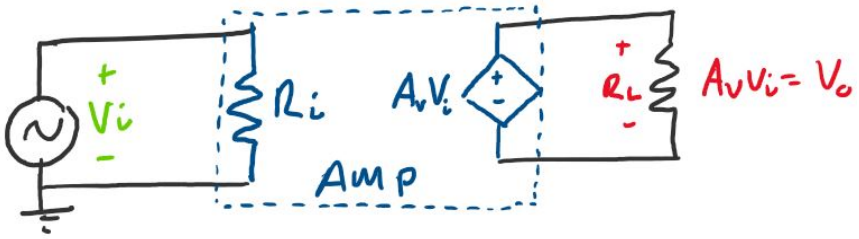
Although we've translated our block diagram to something more closely resembling a circuit, a structure similar to the block diagram remains. On the left, we have our input signal from a voltage source, in the middle our amplifier, and on the right our amplified signal, which passes through a resistor.
As it's common to have a resistor at the output of an amplifier, we give the output resistor a special name: the **load resistor** (R_L). Note that the load resistor can be replaced with any circuit between the output terminals of the amplifier.
Although this model for the amplifier is useful, it's not yet fully developed! We must ask ourselves what's happening *inside* the amplifier block. Thus far, we've considered the amplifier to be a black box - we pass in an input signal, some process happens inside the amplifier, and we get an amplified signal back out.
To analyze amplifier circuits, we must gain a more complete understanding of this process and the circuitry inside the amplifier. To do this, we introduce a new component: the **dependent source**.
So far, our current and voltage sources have operated *independently* of everything else in the circuit.



We recall that no matter what else was happening in the circuit, a DC current source would always put out a constant i_s, and a DC voltage source would always put out a constant V_s.
What if we had a source that *depended* on other values in a circuit? This is what a dependent source is. We may represent a dependent source with the following circuit diagram symbol:



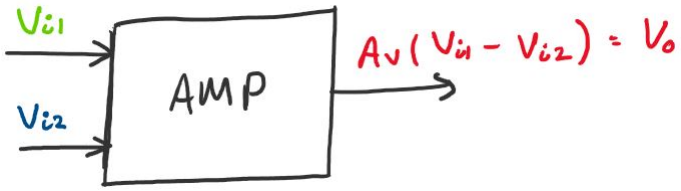
As you can see in the image above, the value of the dependent source, V, is some constant multiplied by another voltage V_o, where V_o is the voltage at some other point in the circuit.
Let's apply the concept of a dependent source to develop a model for what's going on "under the hood" of an amplifier. Using a dependent source, we may represent the amplifier circuit from above as follows:



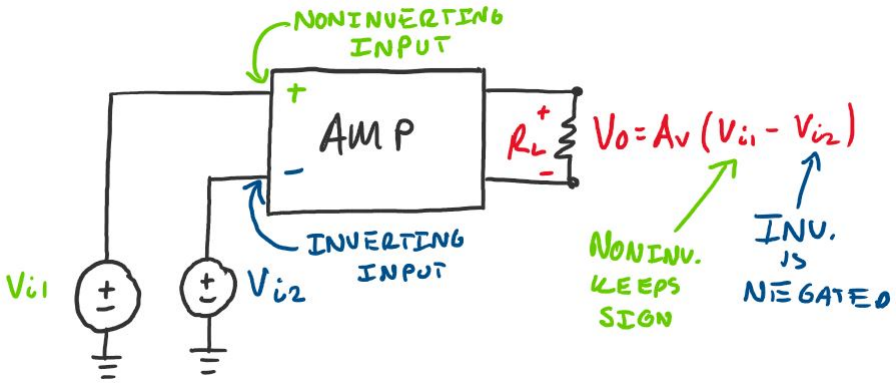
As we can see in the circuit diagram above, we can represent a simple amplifier circuit with two loops. The first loop, on the left, has a voltage source connected to a resistor, R_i, known as the **input resistor**. The second loop, on the right, has a *dependent source* that depends on the value of V_i and amplifies it by some constant A_v, where v stands for voltage. This dependent source then connects to the load resistor outside of the amplifier.

10.1.2 The Differential Amplifier

Now that we've considered a very simple model for an amplifier, with a single input and output port, let's see how we can develop the idea of an amplifier further. Instead of having just a single input, imagine that we have an amplifier with *two* inputs and one output.
The **differential amplifier** is one such device - it amplifies the *difference* between two input signals. A block diagram representing the differential amplifier is found below:



As you can see in the block diagram, two input signals, V_{i1} and V_{i2}, enter the differential amplifier on the left hand side. On the right, a scaled version of the *difference* between the two signals, $A_v(V_{i1} - V_{i2})$, emerges.
Just as we took the step from a simple block diagram representation to a circuit representation for the simple amplifier, we may now do the same for the differential amplifier. We may represent a differential amplifier with the following circuit:

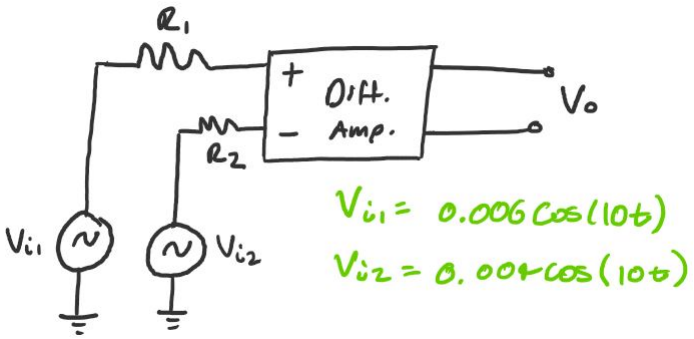


Let's break down what's happening in this circuit diagram. On the left side, we have two inputs to the amplifier, represented by two voltage sources. These inputs then enter the amplifier's input terminals. On the right side, an amplified signal is output across a load resistor. Once again, we label the gain of the amplifier A_v.

Unlike the simple amplifier circuit diagram, we label the two inputs of the differential amplifier $+$ and $-$. We name the input with the positive sign the **non-inverting input**. This is because the signal that enters the non-inverting input keeps its sign in the output signal. We name the input with the negative sign the **inverting input**, as the sign of the signal that enters this port is flipped in the output.

Let's observe how the differential amplifier functions in the following example:

Example: Given the following differential amplifier with a gain of 1000, find and sketch the output signal given the two inputs below:



The key to this problem is in applying the input-output relationship for a differential amplifier. Recall that for a differential amplifier, the output signal may be expressed as:

$$V_o = A_v(V_{i1} - V_{i2}) \tag{337}$$

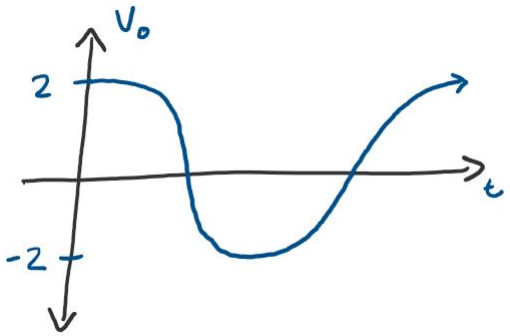
Where V_{i1} is the signal entering the non-inverting input, V_{i2} is the signal entering the inverting input, and A_v is the gain.

Let's apply this relationship to this problem to find what the output signal will be! Looking at the circuit diagram provided, we find that the signal entering the non-inverting input is $0.006\cos(10t)$, and the signal entering the inverting input is $0.004\cos(10t)$. We also know from the problem statement that the gain of the amplifier is 1000. Thus, applying the differential amplifier relationship, we find:

$$V_o = 1000(0.006\cos(10t) - 0.004\cos(10t)) \tag{338}$$

$$V_o = 2\cos(10t) \tag{339}$$

Now, all that remains is to sketch this signal out!



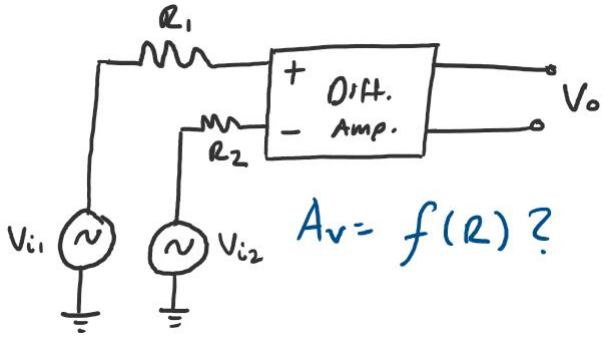
As we can see, by passing signals of *very* small magnitude into an amplifier, we're able to get signals of *much larger* amplitudes yet the same waveform back out.

10.2 Op Amps

Thus far, we've discussed two basic types of amplifiers: the simple amplifier and the differential amplifier. Although, as seen in the previous example, these components *do* allow us to meet our goal of amplifying our signals, they don't allow us to adjust how much we amplify our signals.

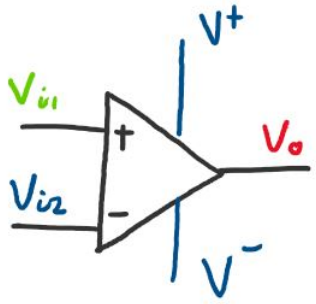
Each amplifier so far has had a single, non-adjustable gain that results in a very rigid amplification. How, then, can we gain *more control* over the level of amplification in our circuits?

Imagine that instead of having a predetermined gain, we were able to *adjust* the gain simply by changing circuit components! For example, imagine that we were able to change the gain of a differential amplifier simply by changing the values of the resistors in the circuit.



Above: What if the gain A_v was a function of resistance?

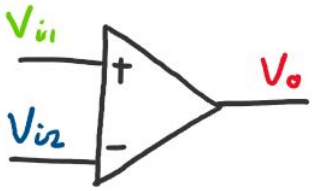
The **operation amplifier**, or **op amp**, is a special type of differential amplifier that gives us this level of control in signal amplification. The circuit symbol for the op amp is the following:[21]



This circuit symbol resembles the conventional layout for a differential amplifier. On the left side, we have two input signals, one of which goes into the non-inverting input terminal and the other of which goes into the inverting input terminal. On the right side of the amplifier, we have the output signal.
Where we notice a difference between op amps and the differential amplifiers we previously discussed is in the two extra terminals, labeled V^+ and V^-, entering the op-amp. These two terminals represent something key to how every amplifier circuit functions.
Thus far, you may have noticed that the amplification of the signal seems to come for free! We pass in an input signal and seem to magically get a much more powerful signal on the other side. Where is the energy for this amplification actually coming from?
The V^+ and V^- ports represent the terminals of an **external power supply**. This external power supply is a high-power source capable of supplying large voltages and currents. *This* is where the amplification truly comes from.
Because we always use an external power supply with an amplifier, it's oftentimes unnecessary to include in the circuit diagram. Thus, in most amplifier

[21] Readers familiar with the program Simulink will recognize the similarity of the op amp symbol to the Simulink gain block.

circuits, we simply *imply* the existence of an external power supply and draw the amplifier without the extra two terminals. An op amp will therefore typically appear as the following in a circuit diagram:



10.2.1 The Ideal Op Amp

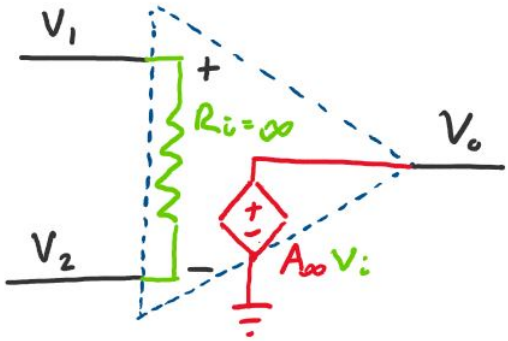
Now that we've established some basic background for the op amp, let's describe the unique properties that afford us so much control over our amplification. An **ideal op amp** has the following surprising characteristics:

1. **Infinite gain**. Whereas the differential amplifiers we've considered in the past have had a high, albeit finite gain, an op amp has *infinite* gain. This means that, using our differential amplifier circuit, any signal passed into an op amp will be amplified to infinity.

2. **Infinite input resistance**. An ideal op amp has an input resistor of infinite resistance between the inverting and non-inverting inputs. This means that *zero current* flows into an ideal op-amp.

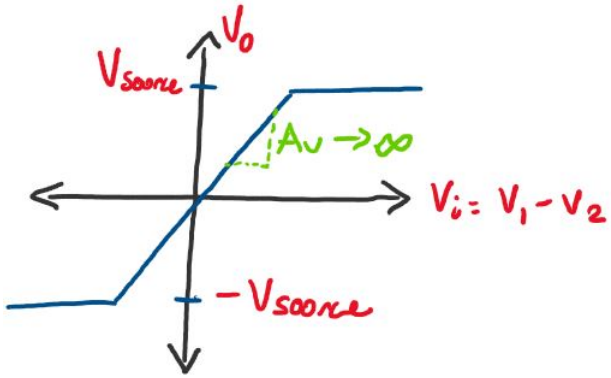
3. **Zero output resistance**. An ideal op amp has zero resistance on the output of the amplifier, meaning that *any current* may flow out of the amplifier.

With these three unusual characteristics in mind, we may draw the following circuit representation of the interior workings of an op-amp:



Above: a circuit representation for the inside of an op amp.

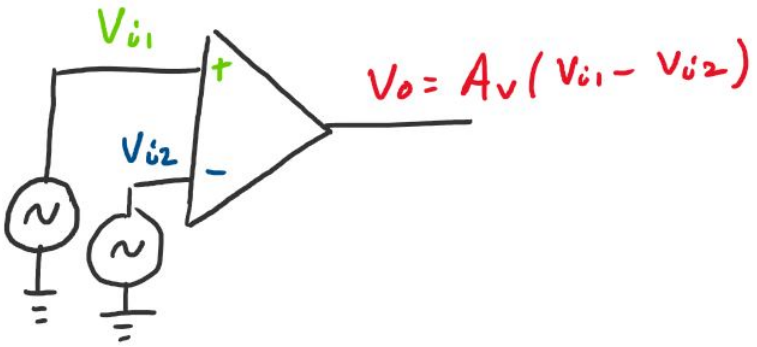
Looking at each component in the diagram above, we see how each of the three ideal op amp characteristics emerges.
First, we notice that the input resistor, R_i, has infinite resistance. By Ohm's law, $V = IR$, we know this means that zero current flows into the input terminals. Second, we notice that the dependent source has an infinite gain, A_∞. Third, we observe that the path from the dependent source to the op amp output is simply a wire, meaning that the op amp has zero output resistance.
The following graph incorporates these three properties, and describes the relationship between the input and output voltages of the op amp:



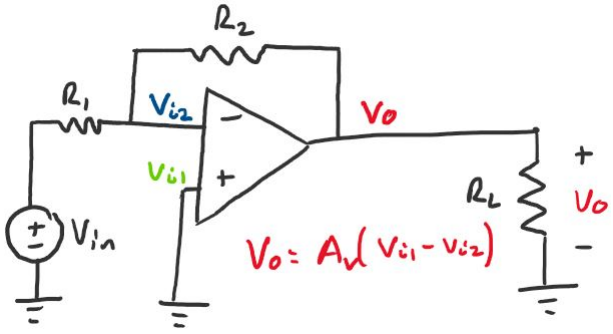
On the x-axis of the graph, we have the input voltage, which, as for all differential amplifiers, is equal to the difference between the two inputs. On the y-axis, we have the resulting output voltage.
In the middle of this graph, we have a linear region of operation, with a slope equal to the gain of the amplifier. For an ideal op amp, this slope approaches a vertical line, since the gain is equal to infinity.
On either end of the linear region, we have two straight lines that seem to "cut off" the output voltage of the op amp - why does this occur?
Recall from our earlier discussion that every op amp is powered by an external power supply. Because of this, the maximum voltage of the op amp output is limited by the maximum voltage of the power supply. Thus, the output voltage of the op amp is never able to exceed the voltage of the source. This results in the sharp, straight-line cutoffs at the voltage of the source.

10.2.2 Negative Feedback

Although all of the characteristics of the op amp, from infinite gain, to infinite input resistance, to zero output resistance, *sound* ideal, we're faced with a problem: how do we actually *use* an op amp?
Let's think about this challenge in terms of the circuit we previously used for a standard differential amplifier:



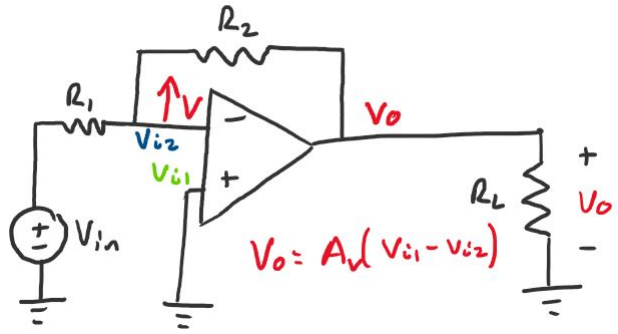
We must now ask ourselves the question: would this circuit work? With an amplifier with a finite gain, for example $A_v = 1000$, we saw that this setup worked just fine and that we received a steady, amplified signal.
But, we know that an ideal op amp has an *infinite* gain. This means that whatever our input signal is, our output signal will have an *infinite* magnitude! Since no external power supply would ever be able to supply this, we're faced with a problem: although we have an amplifier with very desirable characteristics, we have no way to harness its amplification!
Let's examine the properties of the interesting circuit setup below, and see if helps us achieve our goal of harnessing the power of the op amp:



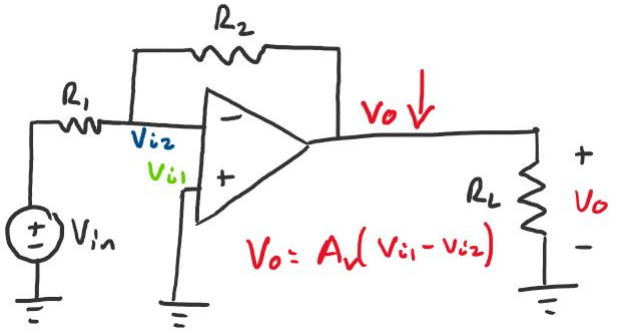
In our analysis of this circuit, it'll be important to keep the input/output voltage relationship for a differential amplifier in mind. Recall:

$$\boxed{V_o = A_v(V_{i1} - V_{i2})} \tag{340}$$

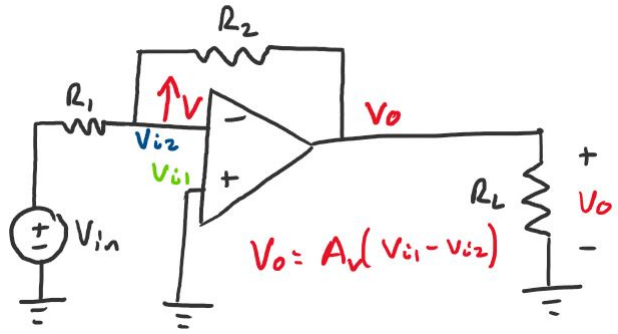
With this relationship in mind, let's begin by imagining we have a *large* positive voltage at our inverting input terminal. We'll indicate this on our circuit diagram with a ↑ as follows:



Let's examine what effect this would have on our output voltage using the differential amplifier relationship. According to the formula, if we have a large voltage at the inverting input, the output voltage will *decrease*. Let's now indicate this with a $\downarrow$ on our diagram.



Looking back at our amplifier relationship, what effect does this have on the rest of the circuit? Since V_o and V_{i2} are connected, this change in V_o will have some effect on V_{i2}. According to the formula, if the output voltage decreases, the inverting input voltage will *increase*. This brings our circuit *back* to the following state:

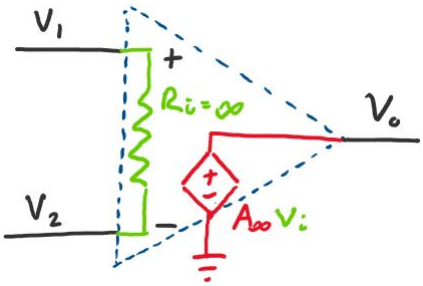


As we can see, the voltage at the inverting input will now increase, starting the cycle all over again. It turns out that through this process of balancing

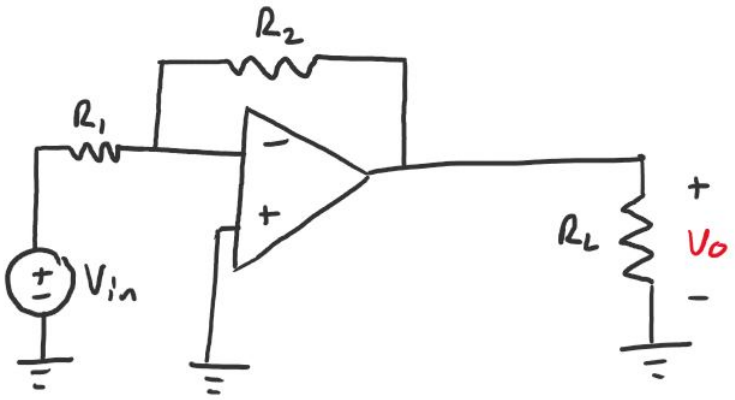
the output and inverting input voltages, the circuit will eventually settle at a *constant* value!
The idea of connecting the output terminal back to the inverting input of the op amp, known as **negative feedback**, is thus the key to getting a constant, non-infinite amplification from an op amp.

10.2.3 Op Amp Analysis

Now that we have a circuit setup we know will allow us to work with an op amp, we must develop the analytical tools we need to solve op amp circuits. Before jumping into an example, let's take another look at the characteristics of an ideal op amp, and see if they offer any hints as to how we may analyze op amp circuits.
Recall the following model for an ideal op amp:



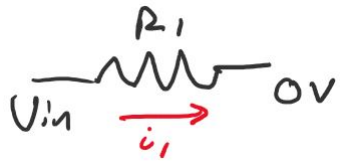
Let's begin with the input resistance. Recall that for an ideal op amp, the input resistance is infinite. This means that however large the input voltages are, the current flowing through the input resistor is 0.
What else can we determine about the input resistor? Let's think back to the input/output relationship for an amplifier, $V_o = A_v(V_{i1} - V_{i2})$. Using this relationship, it can be shown that for the op amp to settle at a constant value, the V_{i1} *must equal* V_{i2}. If this isn't the case, then the op amp will be in an unbalanced, changing stage, similar to those we considered in our discussion of negative feedback.
These two constraints, of the input current to the op amp being zero and the two input terminals being at the same voltage, will prove to be essential to our analysis of op amp circuits.
Now that we've developed the necessary background, we may analyze our first op amp circuit! Let's explore op amp analysis with the following example.
Example: The following circuit is known as an ***inverting amplifier****. Assuming all components are ideal, find the closed loop voltage gain* V_o/V_{in}*.*



Let's break down what this question is asking. We're looking to find the closed loop voltage gain of the circuit. This is the ratio of the output voltage V_o to the input voltage V_{in} when the circuit has a negative feedback loop. Note that this is different to the open loop voltage gain, which would just be the gain of the amplifier, infinity.
When solving op amp circuits with negative feedback, we may use the following procedure:

1. **Verify that the circuit has negative feedback.** Before we begin our analysis, it's important to verify that the circuit we're analyzing actually has negative feedback. If it doesn't, the circuit might not reach a steady value and our analysis will be irrelevant!
   Here, we observe that we have a connection between the output voltage and the inverting input terminals. Thus, we have negative feedback and may continue our analysis.

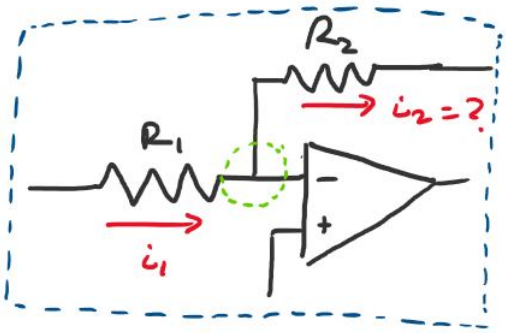
2. **Solve the circuit from left to right using Kirchoff's laws.** Once we know that there's negative feedback in place, we may proceed with solving our circuit. With op amps, our strategy is to start all the way at the left of our circuit, at V_{in}, and work our way over to the right of the circuit to establish a value for V_o.
   Let's begin by solving for the current passing through R_1. We know that the voltage on the left side of R_1 equals V_{in}. What about on the right side?
   To find this voltage, we apply our first op amp constraint. We know that the inverting and non-inverting terminals are always at the same voltage. Thus, since the non-inverting terminal is connected to ground, the inverting terminal *must* be at $0V$. This results in the following:



From here, we may use Ohm's law to find:

$$i_1 = \frac{V_{in}}{R_1} \tag{341}$$

Now that we have i_1, we may continue making our way right in the circuit. Let's now zoom in on the next part of the circuit and calculate i_2, the current passing through R_2.



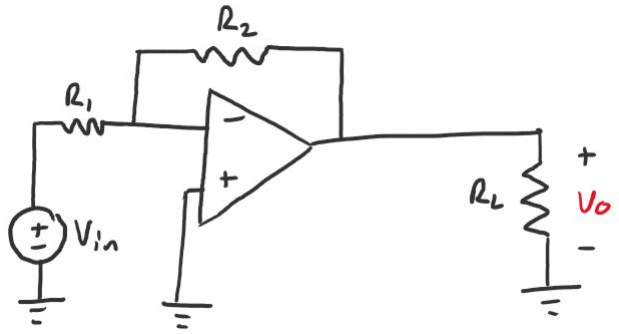
To solve for i_2, we may apply KCL at the highlighted node. We know that i_1 enters the node and i_2 and i^- exit. This gives us the following KCL relationship:

$$i_1 - i_2 - i^- = 0 \tag{342}$$

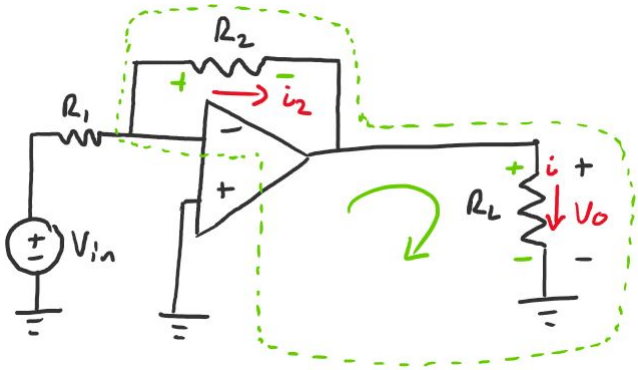
But, we know from our op amp constraints that no current enters the op amp terminals! Thus, $i^- = 0$, and we conclude:

$$i_1 = i_2 = \frac{V_{in}}{R_1} \tag{343}$$

We're now one step closer to finding V_o in terms of the other circuit variables! From here, there are several different ways to solve for V_o. Let's take another look at the circuit diagram to see where we may go next:



We notice that in this circuit, both the inverting input and the end of the load resistor are at $0V$. Since both of these points are at the same voltage, we may make the following surprising KVL loop in our circuit:



Writing out the KVL equation for this loop, we see:

$$V_{R2} + V_o = 0 \tag{344}$$

$$i_2 R_2 + V_o = 0 \tag{345}$$

Substituting in $i_2 = \frac{V_{in}}{R_1}$ and simplifying, we find:

$$0 = \frac{V_{in}}{R_1} R_2 + V_o \tag{346}$$

$$V_o = -\frac{V_{in}}{R_1} R_2 \tag{347}$$

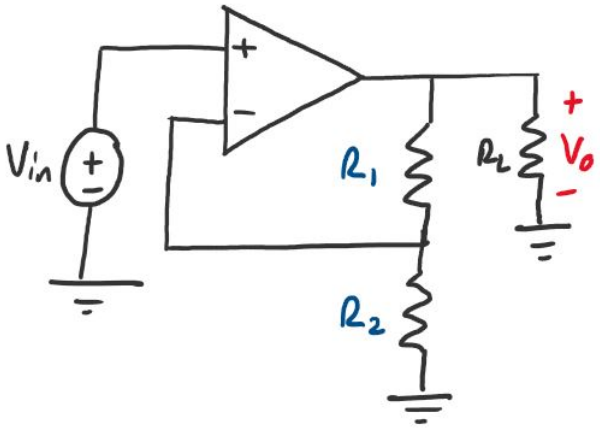
$$V_o = -\frac{R_2}{R_1} V_{in} \tag{348}$$

Recalling that the closed loop gain is the ratio of the output voltage to the input voltage, we divide both sides by V_{in} and reach our final answer:

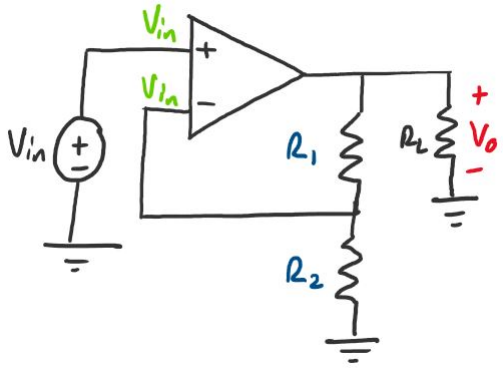
$$\boxed{A_v = \frac{V_o}{V_{in}} = -\frac{R_2}{R_1}} \tag{349}$$

Our problem is now complete!

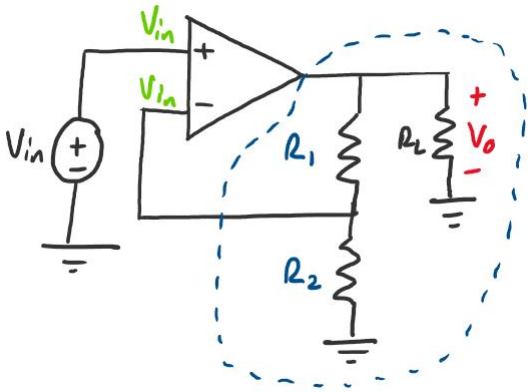
We may now determine several important things about this circuit. Firstly, consider the name of the circuit: the *inverting amplifier*. From our answer, we see that this name comes from the fact that the sign of the output voltage is the opposite of the input! Secondly, we notice that using this inverting amplifier circuit, we're able to control the gain of the amplifier *entirely* through the values of R_1 and R_2.
Thus, with the inverting amplifier circuit, we've managed not only to harness the infinite gain of the amplifier but also to *control* the gain entirely from our circuit components. This highlights the real power of the op amp as a circuit component!
Although the inverting amplifier is certainly a powerful circuit, is there a way for us to get the same control over our circuit and *not* flip the sign of our input signal? To achieve this, we use a circuit known as the **non-inverting amplifier**, which we'll now analyze:
Example: Find the closed loop gain of the following non-inverting amplifier circuit. Assume all circuit components are ideal.



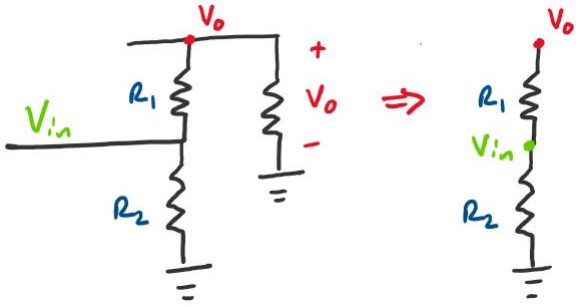
Right away, we notice something different about the non-inverting amplifier from the inverting amplifier. Instead of having both terminals at $0V$, we find that the non-inverting terminal has a voltage source attached to it!
Let's use our op amp analysis procedure to solve this problem. First, we check for negative feedback. Looking at the circuit diagram, we see that there's a path from the output of the circuit to the inverting input terminal. Thus, the circuit has negative feedback and will settle at a constant value! We may now proceed with our analysis.
Starting at the left of the circuit, we note that the non-inverting input is at a voltage of V_{in}. Applying our ideal op amp constraint, we know that the input terminals are always at the same voltage. Therefore, the inverting input terminal is also at V_{in}. We now have the following information about our circuit:



Let's begin to move towards the right in our circuit. Now that we know the voltage of the inverting input terminal, let's focus our attention on the output circuit:



Let's redraw this portion of the circuit and see if we can gain some insight into what's happening:



In this circuit, we know the values of V_{in}, R_1, and R_2, and wish to solve for the output voltage V_o. Looking closer at the circuit above, we recognize that the remaining circuit is actually a voltage divider!

We may thus apply the voltage divider relationship for the output circuit to find:

$$V_{in} = \frac{R_2}{R_1 + R_2} V_o \tag{350}$$

Simplifying this equation further, we get:

$$V_o = \frac{R_1 + R_2}{R_2} V_{in} \tag{351}$$

$$V_o = \left(1 + \frac{R_1}{R_2}\right) V_{in} \tag{352}$$

Finally, dividing both sides by V_{in} to get A_v, we arrive at our final answer for the closed loop gain:

$$\boxed{A_v = \frac{V_o}{V_{in}} = 1 + \frac{R_1}{R_2}} \tag{353}$$

Thus, we observe that with a non-inverting amplifier circuit, we're able to control the amplification of the circuit once again by adjusting the resistances. This time, however, the sign of the output is *not* flipped.

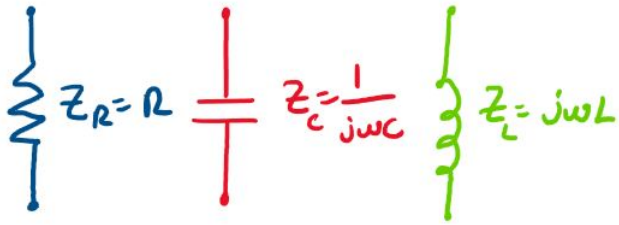
Thus far, we've covered two important types of op amp circuits, and have seen a variety of methods for the analysis of amplifier circuits. One common thread among our analyses, however, has been that all of our sources have been constant. What happens when we use an *alternating* source with an op amp circuit?

Recall that when working with alternating circuits, we use *phasor analysis*, which involves using complex numbers to simplify the process of working with alternating signals.

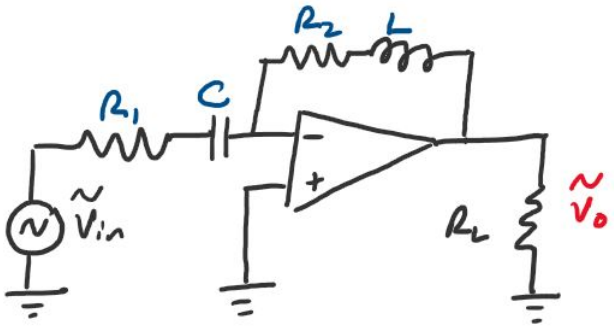
Instead of simply using resistors and resistance, we defined a much more general term called impedance (Z), which we defined to be:

$$\boxed{Z = \frac{\tilde{V}}{\tilde{I}}} \tag{354}$$

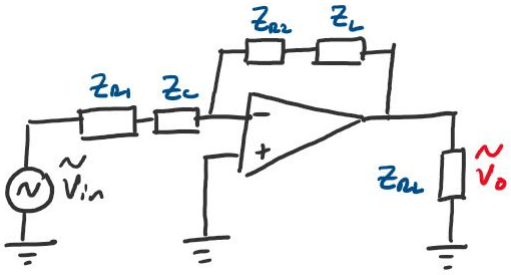
We then proceeded to derive the following formulas for the impedances of resistors, capacitors, and inductors:



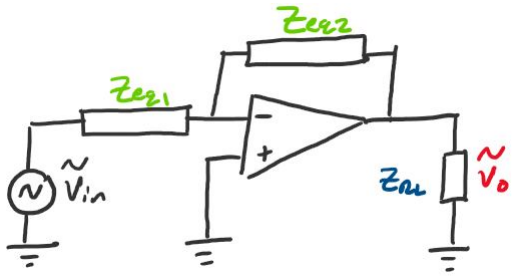
You may remember from our exploration of phasor analysis that all of our DC circuit analysis techniques, such as KCL and KVL, remained the same for phasor analysis.
This same concept applies for op amp circuits. We may solve AC op amp circuits *exactly* as we would DC op amp circuits, simply using impedance instead of resistance when appropriate. Let's apply this idea in the following example:
Example: Find the closed loop gain $\tilde{V}_o/\tilde{V}_{in}$ for the following circuit. Assume all components are ideal.



Let's begin this problem just like we would any other op amp problem! First, we verify that the circuit has negative feedback. Looking at the circuit diagram, we find that there's a path between the output of the op amp and the inverting input terminal. Thus, negative feedback exists, and we're ready to start our analysis.
Let's begin our analysis by completing the first step from phasor analysis. First, redraw the circuit with boxes instead of resistors, capacitors, and inductors, and label each box with an impedance.



Now, let's proceed with our circuit analysis step. Right away, we notice that we can simplify the circuit above using the rules of equivalent impedance. Recall that for components in series, the equivalent impedance is the sum of the individual impedances.
With this in mind, we redraw the circuit as the following:



Where $Z_{eq1} = Z_{R1} + Z_C$ and $Z_{eq2} = Z_{R2} + Z_L$.
Upon closer inspection, we realize that the redrawn circuit is simply an inverting amplifier! Recall from our earlier example that for an inverting amplifier:

$$A_v = \frac{V_o}{V_{in}} = -\frac{R_2}{R_1} \tag{355}$$

Since all of the same rules apply for phasor and DC circuit analysis, we may simply substitute Z_{eq1} for R_1 and Z_{eq2} for R_2, leaving us with:

$$A_v = \frac{\tilde{V}_o}{\tilde{V}_{in}} = -\frac{Z_{eq2}}{Z_{eq1}} \tag{356}$$

Substituting in our expressions for Z_{eq1} and Z_{eq2}:

$$A_v = -\frac{Z_{R2} + Z_L}{Z_{R1} + Z_C} \tag{357}$$

Now, all that remains is to plug in the impedance of each component:

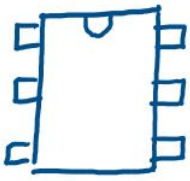
$$\boxed{A_v = -\frac{R_2 + j\omega L}{R_1 + \frac{1}{j\omega C}}} \tag{358}$$

And our problem is complete!

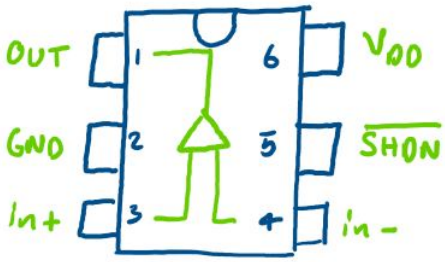
10.2.4 The Real Op Amp

In describing all of our circuits, in using Ohm's law for resistors to the simple differential equations for capacitors and inductors, we've relied on the fact that our circuit components are *ideal*. By using ideal models for circuit components, we're able to analyze circuits quickly and come to useful conclusions using relatively simple equations.
After describing the circuit diagram and characteristics of an ideal op amp, it's natural to ask: what does a real-world op amp look like?
In reality, op amps are constructed from circuits *far* more complicated than the simple input resistor/dependent source circuit we studied. Real op amps are typically constructed from around 30 transistors, as well as a set of resistors and capacitors.

Since the circuits used to construct op amps are so complex, real-world op amps are built using **integrated circuits**. Integrated circuits, or "ICs," are chips that contain tiny, complex circuits that are otherwise hard to build.
A typical op amp IC will look similar to the following:



The rectangular block in the center contains all of the tiny transistors, resistors, and capacitors within. This is where the actual op amp is. On the outside of the rectangle, we find metal terminals that can be used to interface with the op amp inside.
How does this integrated circuit relate to the circuit diagram for an ideal op amp? The following is the setup for the TLV2370, an op amp manufactured by the company Texas Instruments.[22]



As you can see in the image above, all of the terminals of the integrated circuit correspond to the different ports of the op amp circuit we previously discussed. The inverting and non-inverting inputs may be found in the bottom row of pins, while the output of the op amp is found in the top left. The external power supply for the op amp, labeled V_{DD}, is found in the upper right.
By connecting each pin correctly, we may use an integrated circuit op amp just like any other circuit component!

[22] The name Texas Instruments (TI) may be familiar to you as the manufacturer of your calculator! TI is also a large manufacturer of electronic components.