# Build Cool Stuff With Python

By Doug Purcell

Website: [http://www.purcellconsult.com](http://www.purcellconsult.com/)

Preamble I: Python Install

Preamble II: PyCharm Setup

Preamble III: Python Crash Course

-------------------------------------------

Project #1: Your Biography

Project #2: Converters

Project #3: Number games

Project #4: Rock Paper Scissors

Project #5: Fantasy text based game

Project # 6: Word guessing game

Project #7: Practical computer art

Good article about learning programming: [https://www.reddit.com/r/learnpython/comments/bhccr1/i\_didnt\_know\_anything\_about\_programming\_three](https://www.reddit.com/r/learnpython/comments/bhccr1/i_didnt_know_anything_about_programming_three/)

# General Guide updates and ongoing mentoring

# Still stuck and confused about taking your python skills to the next level? Get in touch with me through the following mediums:

# Subscribe to the slack channel: [https://purcellconsult.slack.com](https://purcellconsult.slack.com/messages/)

# Add me on Skype:purcellconsult

# Subscribe to my newsletter: <http://www.purcellconsult.com/newsletter>

# Preamble I: Python Install

## Install Python on Windows

It’s a high probability that if you’re running a Windows operating system then Python won’t be there by default. To discover if Python is installed on your machine you can open the terminal and then type python. If it’s installed then that command will run *python.exe* and reveal the version number. If you get a rude message like the following:

'python' is not recognized as an internal or external command, operable program or batch file

This tells you that Python is not installed and you have to set it up. Follow the steps below to install and setup Python on your computer.

**Step one**: Download the [latest version of Python](https://www.python.org/downloads) on your machine:

**Step two**: Open and start the Windows installer that matches your system. If you click “Install Now” then Python is installed in the “user” directory, but if you change its location then make a note of where it’s installed.

**Step three**: You’ll have an option to add Python to PATH which is where the computer searches for Python when you type it via command prompt. If you check this box then Python will be available via this option, if not then when you type *python* in the console an error will occur. Therefore, it’s a good idea to check this option so that you can type in python commands via command prompt. If you installed Python without selecting this option then no biggie as you have to manually add the path to your system. Below are the steps:

* In the Windows menu search for advanced system settings and select  
  view advanced system settings.
* In the window that displays click *Environment Variables*.
* In the next window, find and select the user variable called path and click *Edit*.
* Scroll to the end of the value and add a semicolon (;) followed by the location of *python.exe*. If you didn’t change the default installation location it should be located in your user directory.
* Click OK to save the settings

If you don’t know the location of python.exe then don’t panic, just search for *python.exe i*n the Windows menu. Once located, right click the file, select properties, and view the Location. Right click to copy the full path and then paste it at the end of the Path user variable. If you don’t have a Path user variable then click the new button, add a variable named Path, and then add the value which is the location or “path” of the python.exe file. Once done type “python” into the terminal to ensure that everything was set up properly and that it runs.

## Install Python on OS X

Like Linux, Python is already installed on a variety of OS X systems. You can confirm that Python is installed by going to: *Applications → Utilities → Terminal*. Next, type the following into the terminal:

python -V

The command will output the version of Python which is:

Python 2.7.3

Any version between 2.7.0 and 2.7.10 is common. The next step is to test if you have Python3 on your computer. You can do this by typing the following into the terminal:

python3

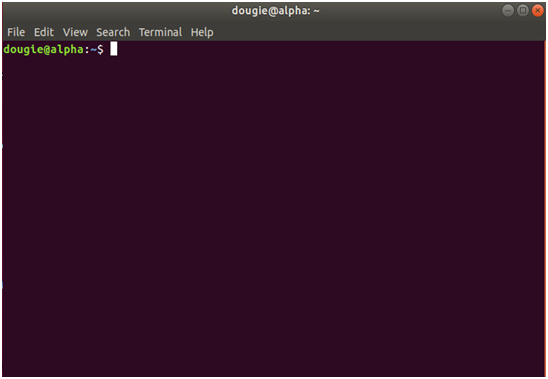
If the output shows that Python 3 is installed then you’re safe… for now. If you get an error then that’s not cool and you have some work to do. You can fix this by downloading and installing Python with the [appropriate Mac OS X installer](https://www.python.org/downloads) that matches your system.

### **Installing Python on Linux (Ubuntu 18.04)**

To see if Python is installed on your machine open up the terminal and type in the following:

python

You can fire-up the terminal by using the keyboard shortcut: *ctr + alt + t*. The terminal in Ubuntu 18.04 looks like the following:



###### *Figure 1.0: Install Python on Linux.*

The output should look something like the following:

Python 3.6.5 |Anaconda, Inc.| (default, Apr 29 2018, 16:14:56)

[GCC 7.2.0] on linux

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

Look at this line of output:

Python 3.6.5 |Anaconda, Inc.| (default, Apr 29 2018, 16:14:56)

If you got something like this then *woot-woot*, Python 3.6.5 is installed on your machine. If Python 2.7 or later is installed then it’s OK, you don’t need to uninstall it, you just need to get Python3 running. Luckily this process is super easy with Ubuntu:

* Step one: Open up the terminal by pressing ctr + alt + t
* Step two: Type sudo apt-get update
* Step three: Type sudo apt-get install python3.6

The word sudo is abbreviation for “super user do” and it allows programs to be executed as a super user, aka the root user. The apt command means Advanced Package Tool, which is a package manager for Debian based operating systems like Ubuntu. The apt-get command is the APT package handling utility. You can see a list of the commands that’s available for it by typing *apt-get* into the terminal.

A short term alternative is to use an online python interpreter. Here’s some of the following:

* Online GDB: <https://www.onlinegdb.com/online_python_interpreter>
* Repl.it: <https://repl.it/languages/python3>
* Another online python interpreter <http://mathcs.holycross.edu/~kwalsh/python>

## Preamble II: Pycharm Setup

There are many choices of integrated development environments (IDEs) in python that you can choose from such as sublime, IDLE, Vim, Wing, Atom, Spyder, and PyCharm. There’s way too many for me to keep track of so you can always check out Wikipedia: <https://wiki.python.org/moin/IntegratedDevelopmentEnvironments>

If you have python installed, have an editor or IDE you’re comfortable with, and ran the standard *Hello World* program then you can jump to *preamble III*. If you don’t already have an editor or IDE then I’ll recommend PyCharm which has the largest mind share in the python community.

Before you download and run PyCharm you need to have the python interpreter installed. There’s three flavors to PyCharm which are the community, education, and professional edition. The community edition is open source and compared to the commercial version (professional), it comes equipped with less features. However, when you’re starting programming the community edition will suffice. Below are the instructions on the various operating systems:

* **Windows:**
  + Download the PyCharm installer, run the executable file, and follow the wizard steps. Here’s the instructions on the PyCharm website: <https://www.jetbrains.com/help/pycharm/installation-guide.html?section=Windows>
* **MacOs:** Download the PyCharm disk image and mount and drag the image to the Applications folder: <https://www.jetbrains.com/help/pycharm/installation-guide.html?section=macOS>
* **Linux**: If you have Ubuntu 16.04 you can install PyCharm through the command line using the snap package manager. $ sudo snap apt-get install pycharm-community

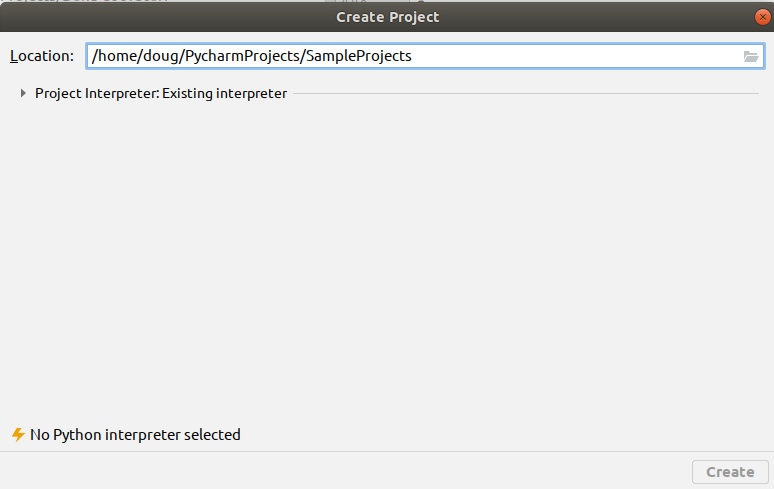
## Hello World With PyCharm

# Once you’ve installed PyCharm the next stage is to run the proverbial Hello World program. Here’s the step by step procedure on how to do this.

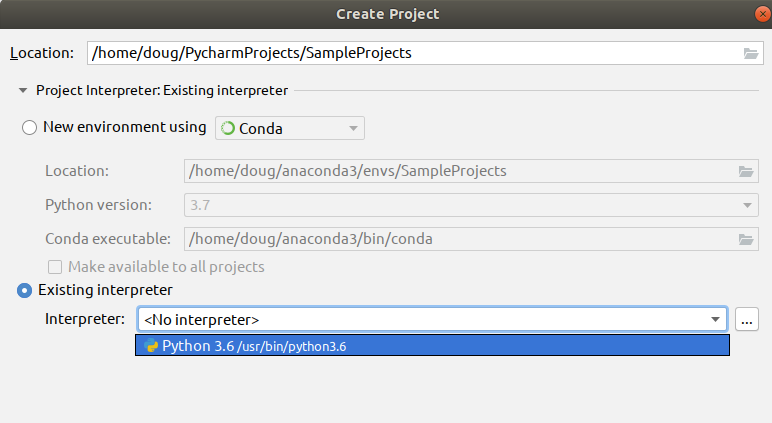
### Create a Fresh Project

Create a new project by doing the following: *File → New Project*

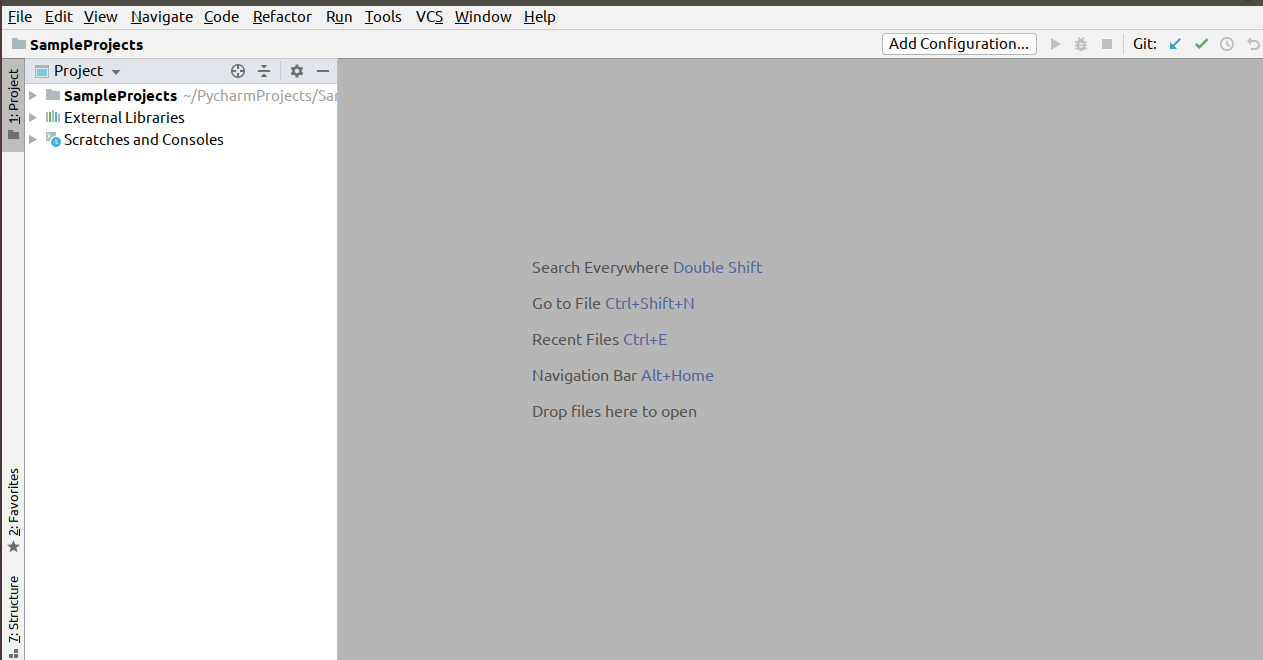
Name the project *SampleProjects*. A project is an organizational unit in PyCharm. Here’s a screenshot of what the setup should look like thus far:

****

You need to select the python interpreter you’re using before PyCharm if one is not already selected. Click the arrow that’s next to *Project Interpreter:Existing Interpreter.* Select a python 3.0+interpreter. Below is an example of how my setup looks:



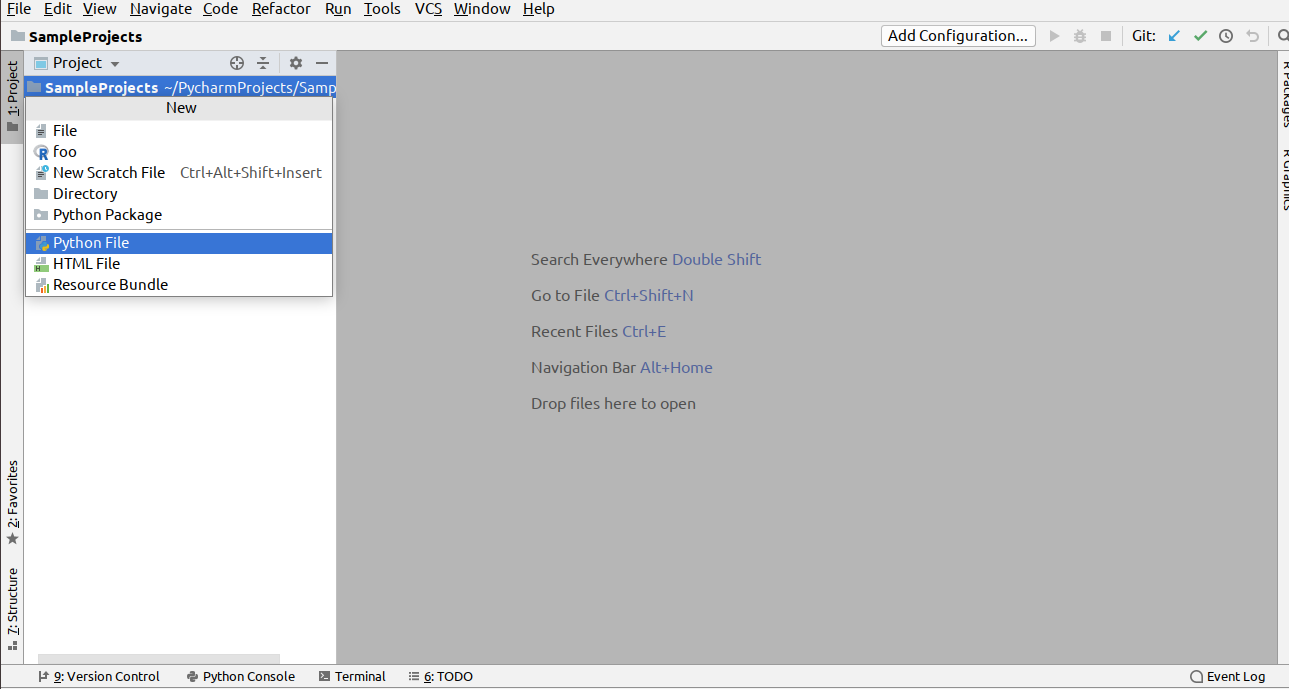
Once the python interpreter is selected click the create button to create your project. Here’s a screenshot of the setup:



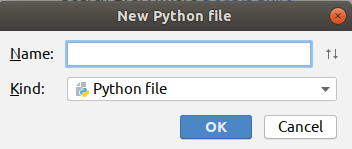
### Create a fresh python file

At the main menu on PyCharm, or the portion where you see the various menus such as File, Edit, and View, do the following:

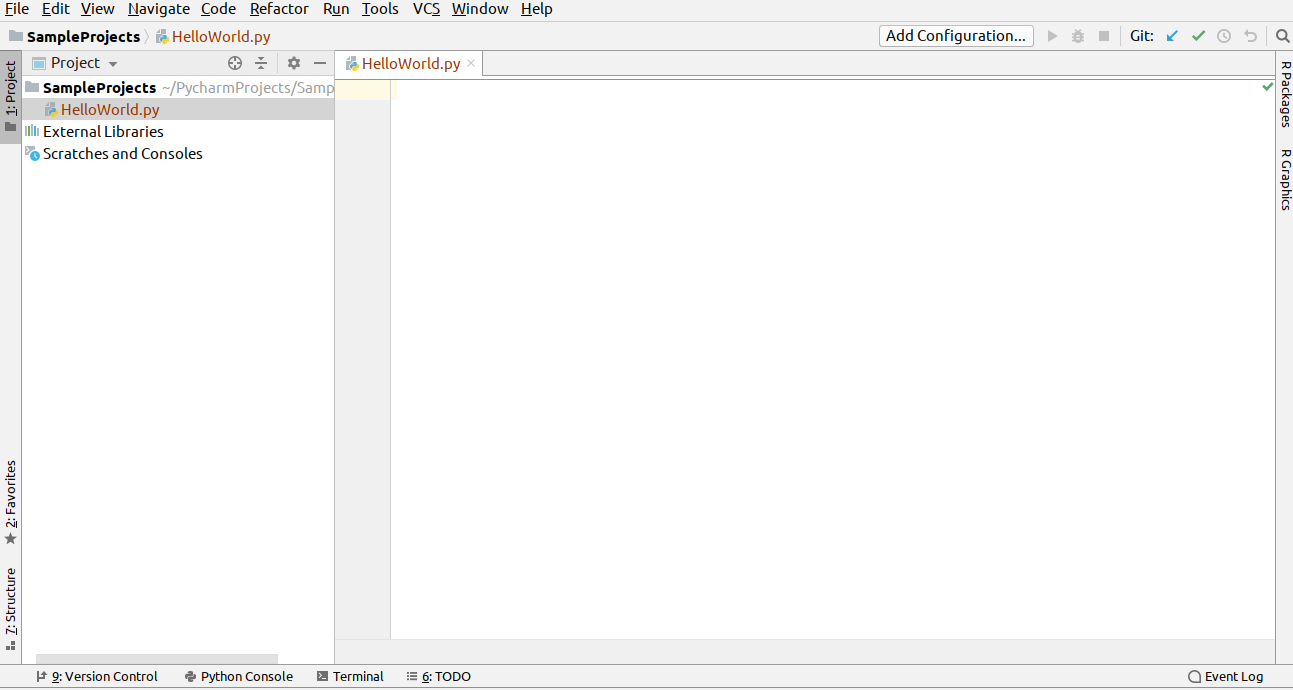
**File → New File → python file.**



A text box should open which looks like the following:

**

Enter in the name *HelloWorld* and select the OK button. Once done here’s how your project setup should look:



The blank white space is known as the editor. That’s where you’ll spend most of your time hacking away. The left hand side is known as the project manager, that’s where you can see the organization of files in a project.

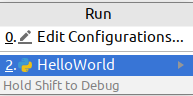
### Add code and run the file

Copy the following code into the editor:

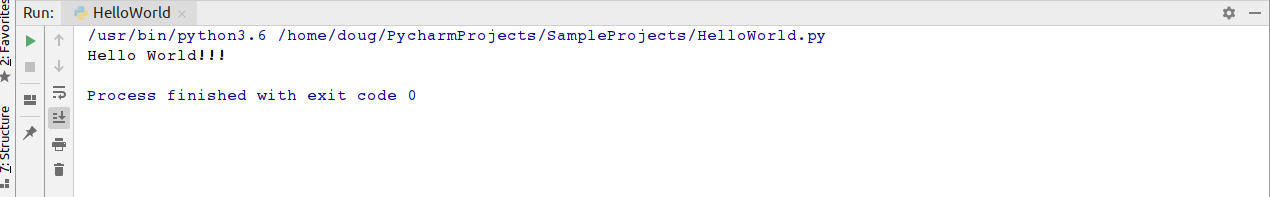
print(**'Hello World!!!'**)

To run the file click the following on the main menu: run → run

A dialog box should popup which looks like the following:



Select the *HelloWorld* program. Once the program has executed you should see the text *Hello World!!!* in the console as indicated in the screenshot:



If all went well then congrats you’ve ran your first python program. Now, there’s already a plethora of free curated information about PyCharm online. Here’s a couple of places to whet your appetite:

* Quick Start Guide: <https://www.jetbrains.com/help/pycharm/quick-start-guide.html>
* PyCharm blog: <https://blog.jetbrains.com/pycharm>
* Learn keyboard shortcuts for editing, navigating, refactoring, and debugging: <https://www.jetbrains.com/help/pycharm/mastering-keyboard-shortcuts.html>

## Preamble III: The 90 Minute Python Mini guide

You can learn the gist of python in 90 minutes. It will be a superficial level, but it’s a start nonetheless. On April 3rd 2019 I did a tech presentation with the goal of teaching the gist of python in 45 minutes. I’ve extended that material into this mini guide. If you want to gain a deeper understanding of the basics of python then I’ll recommend checking out another book of mines titled *Become a Python Developer:* [*https://www.amazon.com/Become-Python-Developer-Wrestle-Defeat-ebook/dp/B07KX8RT4V*](https://www.amazon.com/Become-Python-Developer-Wrestle-Defeat-ebook/dp/B07KX8RT4V)

Let’s get to it...

## Variables

A variable in python is similar to a variable in mathematics. It’s something that has a changeable state. Examples of variables in python are shown below:

a = 10

b = 1.598

c = .1987

d = 100.579

## Printing output

The above code snippet simply stores the variables in the computer’s memory. This means that the data is there but you as the user can’t see it. In order to view the data you need to use the print() function to display the output as shown below:

print(a)

print(b)

print(c)

print(d)

## Swapping variables

A useful tip to know in python is that to swap variables you can do that in a single statement.

x = 5

y = 10

z = 30

x, y, z = z, x, y

print(x)

print(y)

print(z)

The output will be:

30

5

10

## Variable Naming Tips

For details on how to properly name variables in python, refer to the python enhancement proposals also known as PEP 8.

Here are some of the highlights of PEP 8:

# Variable names can have letters, numbers, and underscores.

# Can't use a reserved word like 'print'.

# Be as descriptive as possible with your variable names. This reduces ambiguity and helps make your code more maintainable when other developers follow in after you.

# Python IS case sensitive so apple is not the same as Apple.

# Put constants, or variables that value is fixed in all CAPS. I.e, DAYS\_OF\_WEEK = 7

For a more comprehensive overview refer to PEP 8: <https://www.python.org/dev/peps/pep-0008>

## Python Math Operators

What is computation? It’s the action of mathematical calculation. The word computation has computer in it, which gives a hint to one of the uses of computers, which is computation. Python like many sophisticated programming languages can be used as a *souped up calculator*. All of the standard features that’s available on scientific computers can easily be emulated with the help built in operations and modules in python. Let’s look at some of the math operators available in python: +, -, \*, \*\*, / , //, %

Most of these symbols you’re probably already familiar with. Let’s dig into some code to better understand this:

print(10 + 10)

print(50 - 10)

print(10 \* 10)

print(20 \*\* 2)

print(9 / 5)

print(8 // 3)

print(11 % 5)

print(1e10)

Here’s the output:

20

40

100

400

1.8

2

1

10000000000.0

The +, -, \*, and / symbols behave as we expect. The asterisk means multiplication and forward slash means division. The double star, (\*\*) means raise to the power. So, in this case 20 \*\* 2 means 20^2 or 400. The double forward slash (//) indicates the floor operator in mathematics. This means to divide the dividend by the divisors, and ignore the remainder. In this case this means 8 divided by 3, which is 2.66666666667, but the floor operator means to ignore all of the stuff that follows the decimal point in this case the mantissa, so the answer is 2. The % sign indicates modulus, so you divide the dividend by the divisor like you would with regular mathematics except you take the remainder. Therefore, you do 11 divided by 5 which is 2, and then take the remainder which is 1.

## Adding additional functionality into your programs

While we could make use of the builtin mathematical operations, if we use just them then we’re severely limited in what we can do for our mathematical calculations. Luckily there’s the math module which you can checkout here: <https://docs.python.org/3/library/math.html>

It includes mathematical properties like logarithms and trigonometry. To use these functions in your program you need to use the import statement. Below is a quick example:

import math

print(math.log(1000000, 2))

print(math.sqrt(9))

print(math.cos(100) + math.sin(90) + math.tan(90))

print(math.pi\*\*2 \* math.e)

19.931568569324174

3.0

-0.23888487632000044

26.828366297560617

## Strings

If you have ever sent a SMS text, used Facebook chat, or sent an email then you have used strings. A **string** is a sequence of characters wrapped in quotes; in python it could be a single, double, or triple quotes. The single or double quotes can be used interchangeably. The triple quotes are typically used as doctrings, or comments inside methods, functions, or classes. They’re typically used when you need to include text that expands multiple lines as they can handle line breaks nicely. The best way to understand the difference between the various strings type is to create a simple python program and experiment with them. Below is a quick overview of strings in python:

city = 'Los Angeles'

# indexing: python is a zero based indexed language

print(city[0]) # L

print(city[3]) # empty space is a string!

print(city[4]) # capital A

print(city[-1]) # negative indices are permitted

# len() function: gets the length of the string

print(len(city)) # 11

print(city[len(city)-1]) # s

# concatenation: the combining of multiple strings

print('john ' + 'doe ' + 'public') # john doe public

# slicing: retrieves ranges of a string

print(city[0:3]) # Los

print(city[4:11]) # Angeles

print(city[::]) # Los Angeles

print(city[::2]) # LsAgls

print(city[::-1]) # selegnA soL

## Boolean Algebra

This is a branch of mathematics that was invented by English mathematician George Boole back in 1847. Even though it’s over a century old it’s impact still persists. It has been fundamental in the development of digital electronics, and is available in all modern day programming languages. Therefore, learning Boolean algebra for python means that you can apply that set of logic to a wide array of languages like Java, C++, Haskell, Erlang, or R.

In python what you need to worry about is the truth values of true or false, which are typically denoted by 1 or 0 respectively. The main operations you need to worry about are and (conjunction), or \*disjucntion), and negation. There’s also the lesser used *xor* operator.

**Below is a sample of how the truth table looks**:

x y x and y x or y x not x

0 0 0 0 0 1

1 0 0 1 1 0

0 1 0 1

1 1 1 1

Remember, 0 maps to False, and 1 maps to True. A shortcut to remembering this is and is always False or less you have two True operands. Or, is always *True* or less you have two *False* operands. If you’re confused about this no worries, just commit the above truth table to memory. You’ll need to remember it in order to to do conditionals.

is\_the\_sky\_blue = True

do\_cats\_bark = False

print(is\_the\_sky\_blue) # True

print(do\_cats\_bark) # False

Remember, 0 and 1 could interchangeably be used for False or True respectively. Therefore you could used them interchangeably if you desire, even though True or False are more commonly used.

## The 'and' truth table

The 'and' operator evaluates to false in all situations except when both operands are False.

print(True and True) # True

print(True and False) # False

print(False and False) # False

print(False and True) # False

## The ‘or’ truth table

Or evaluates to true with at least one true operand

print(True or True) # True

print(True or False) # True

print(False or False) # False

print(False or True) # True

## The ‘xor’ truth table

Xor is a little tricky. It evaluates to true when the two operands are different.

print(True ^ True) # False

print(True ^ False) # True

print(False ^ True) # True

print(False ^ False) # False

## Control Flow in Python

Once you understand Boolean algebra you can apply that newfound knowledge to control flow in python. Control flow allows you to control the order in which statements are executed in python. There’s the if, else, and elif statements that helps you to control this in python.

## if/else statement

Here’s an example of an if/else statement:

x, y, z = 5, 10, 15

if x < y and z > y:

print(x)

else:

print(y)

The if keyword is a reserved keyword in python, and the expression most be terminated by a colon. If the first expression is true then the statements inside the body are executed, if it’s false then the branch under the else statements are executed.

## elif statement

Below is an example of the elif statement in python:

from random import randint

# picks a random number in range 1...100

grade = randint(1, 100)

if grade >= 90 <= 100:

print('A')

elif grade >= 80 <= 89.9:

print('B')

elif grade >= 70 <= 79.9:

print('C')

elif grade >= 60 <= 69.9:

print('D')

else:

print('F!')

## Ternary Statement

Is a special type of operator that evaluates something based on a condition being True or False. The best way to understand it is to take a look at a simple code snippet:

mood = True

state = 'nice' if mood else 'not so nice'

print('state = {}'.format(state))

The following prints nice because if mood evaluates to True.

## Comments

At this point you may have saw the hash symbol (#) followed by text. This is known as a comment in python and this portion of the code is ignored by the interpreter. However, it’s still very useful to include in your programs as it helps other programmers that may be messing around in your code to understand the logic.

*# This is a comment*

*# This is a comment and will be ignored by the interpreter*

*# I think you get the memo!*

# Iteration in Python

Iteration is the process in which computers do repetitive tasks. Humans despise repetition while computers are amazing at it. Humans can do repetitive tasks like summing all of the numbers from 1-100 manually (assuming no mathematical formulas are used), but these tasks are tedious and error prone. Computers can do number crunching like this in very quick times, like in a couple of nanoseconds. The two main ways to do iteration in python is by using either the while or for loops.

## WHILE LOOP

A while loop states that while a condition is true to execute the statements in the body.

# sets while loop starting at

i = 0

# condition

while i < 10:

print('i = {}'.format(i, end=' ')) # print value of i, end='' means print on same line

i += 1 # increment i

The above prints 0 … 9.

# Sum numbers from 1...1000 in nanoseconds

i, sum = 0, 0

while i < 1000:

i += 1

sum += i

print('The summation of 1...1000 = {}'.format(sum))

## for loop

Another way to iterate in python. It can be used with the range() function to iterate over a sequence of numbers or it can be used standalone to iterate over data structures like lists or sets. Below is a simple example of a for loop in python:

for x in range(10):

print(x, end=' ')

The above prints the numbers 0 … 9 on the same line separated by a space.

## Fibonacci numbers

The following prints the 12th Fibonacci number:

x, y = 0, 1

for z in range(10):

next = x + y

x, y = y, next

print('12th fib number = {}'.format(next))

## Creating functions in python

A function is a set of inputs that map to a set of outputs. You can create your own functions in python by using the *def* keyword.

def scale\_number(num, amount):

return num \* amount

print(scale\_number(10, 5))

Outputs 50.

## Keyword arguments

def area\_triangle(height=11, width=7.5):

return 1/2 \* (height \* width)

print(area\_triangle())

41.25

print(area\_triangle(height=20, width=100))

1000.0

## Accepting an arbitrary number of input

You can do this by attaching an asterisk in front of the variable.

def multiply(\*args, y=1):

for x in range(len(args)):

y \*= args[x]

return y

print('multiply=', multiply(1, 2, 3, 4))

...

*multiply= 24*

## Reading in an arbitrary number of keyword arguments

You can accomplish this by using two asterisks in front of the variable name.

def key\_value(\*\*kwargs):

for key, value in kwargs.items():

print('{} {}'.format(key, value))

key\_value(a=5, b=10, c=15)

...

*a 5*

*b 10*

*c 15*

## Classes and Objects

Object oriented programming is a style of programming that evolves the heavy use of classes and objects. Classes are typically described as blueprints, while objects are described as the templates that’s created from the classes. Below is a simple example of how to create a class in python:

class Point:

"""Simple class in python. This is an example

of a docstring, or a string that's used like a

comment to document a segment of code."""

def \_\_init\_\_(self, x, y):

self.x = x

self.y = y

def get\_x(self):

return self.x

def get\_y(self):

return self.y

def set\_x(self, new\_x):

self.x = new\_x

def set\_y(self, new\_y):

self.y = new\_y

def get\_point(self):

return self.x, self.y

p = Point(5, 10)

print()

print(p.get\_point())

p.set\_x(100)

p.set\_y(200)

print(p.get\_point())

...

*(5, 10)*

*(100, 200)*

# Data structures in python3.6

There are four builtin data structures in python which are lists, tuples, dictionaries, and sets.

## list are mutable collection of objects

Below is a demo of a list in python:

evens = [0, 2, 4, 6, 8, 10]

# reverses the list

evens.reverse()

*...*

*[10, 8, 6, 4, 2, 0]*

# adds an object to the list

evens.append(100)

*...*

*[10, 8, 6, 4, 2, 0, 100]*

# merges another list with the list

evens.extend([1, 3, 5, 7, 9])

*...*

*[10, 8, 6, 4, 2, 0, 100, 1, 3, 5, 7, 9]*

# pops an item from the list

*evens.pop()*

*9*

*...*

# iterating over a list

for x in evens:

print(x)

...

10

8

6

4

2

0

100

1

3

5

7

## Tuples

Are an immutable sequence. Unlike lists once you create a tuple they cannot be modified. Trying to do so will cause an error.

nums = (1, 3, 5, 7)

print(nums)

...

*(1, 3, 5, 7)*

## Dictionaries

These are key/value pairs, or associative arrays in some languages.

vowels = {'a': 0, 'e': 0, 'i': 0, 'o': 0, 'u': 0}

for key, value in vowels.items():

print(key, value)

*...*

*a 0*

*e 0*

*i 0*

*o 0*

*u 0*

## Sets

Stores unique items.

letters = {'a', 'a', 'a', 'b', 'b', 'b'}

print(sorted(letters))

...

*['a', 'b']*

## Advance topics

There’s some advance topics in python that you can learn that will help you when you’re start building more interesting and complicated projects. Some of these features include exception handling, decorators, meta classes, magic methods, generators, and C extensions.

## Exception handling

There’s at least two distinguishable types of errors: syntax and run time. Exceptions occur when the program is being ran and you can handle them by using try/except statements. Here’s a demo of a simple try/except statement in python:

def divide(num, den):

try:

x = num / den

print('{} / {} = {}'.format(num, den, num / den))

except ZeroDivisionError:

print("can't divide by zero.")

divide(10, 5)

divide(0, 10)

divide(10, 0)

## ...

10 / 5 = 2.0

0 / 10 = 0.0

can't divide by zero.

The statement that’s tried to be executed is located in the try block. If an error occurs during the try block then the except block is executed. *ZeroDivisionError* is one of the many builtin exceptions in python3. To view a list of built in exceptions read the python docs here: [insert link to builtin exceptions]

Below is another example of a try/else/except statement in python:

**def** import\_test():

**try**:

**import** math

**import** operating

**import** sys

print(math.pi)

print(sys.version\_info)

**except** ImportError:

print(**"Couldn't import something"**)

import\_test()

...

Couldn't import something

The reason for this is because operating is not a builtin module in python and therefore an error was triggered while in the try block. You can also use the raise statement to force an error to happen. Below is an example of an example of this in action:

try:

a = input('Enter an integer ')

raise Exception("Something strange happened")

except ValueError:

print("An exception happened.")

Enter an integer 10

Traceback (most recent call last):

File "<stdin>", line 3, in <module>

Exception: Something strange happened

Below is an example of a try/except/finally statement:

**def** divide(a, b):

**try**:

result = a / b

**except** ZeroDivisionError:

print(**"Can't divide by 0"**)

**else**:

print(result)

**finally**:

print(**'This is in the finally statement'**)

divide(10, 2)

5.0

This is in the finally statement

...

The finally clause is executed before leaving the try statement. It’s always executed, no exceptions (no pun) regardless if an exception occurs or not.

## Nested functions and decorators

A nested function is a function inside another function. It’s difficult to understand a popular feature in python, decorators without first understanding nested functions and how they work.

**def** outer():

*"""this is outer"""*

x = 5

print(x)

**def** inner():

*"""This is inner"""*

x = 10

print(x)

inner()

In the above code snippet, the outer function declares a variable x and prints it. In the inner function, another variable is created and printed. The inner() function is called within the outer function. When the outer function is called then the value of x inside of outer() and inner() are displayed. Here’s a question. What if inner() was never called? Would 10 still print? Modify the code and see what happens. Here’s something important that you should just commit to memory:

Everything in python is an object.

If everything in python is an object then this means that functions are also objects. Then, that means we can assign functions to variables the same way that we can assign other objects like ints or strings to variables. Let’s play with some code:

**def** a(h):

*"""outer function"""*

x = h + 5

**def** b():

*"""inner function"""*

**return** x \*\* 2

**return** b()

The above contains an outer function and that includes a single statement. The inner function returns a value from the outer function squared. Then, in the final statement the function is returned. Going back to the statement that everything in python is an object then this means that returning a function (an object like an int) is perfectly legal.

c = a(5)

print(c)

The above statement prints 100. In the outer function x = 10. Then, the inner function returns x \*\* 2 which is 10. The inner function can access the values of the outer function. This is why the inner function is also referred to as *wrapper* functions.

Next, is the critical step of returning b. If b() is not returned then that doesn’t mean an error will occur. But, we will not get 100 when c is printed. Update the code to see what happens.

Now that we got some experience with nested functions the next step is to look at decorators. Here’s a quick example:

**def** sprinkle(func):

**def** wrapper():

func()

print(**'This is the decorator in action'**)

**return** wrapper

@sprinkle

**def** im\_mute(): *# we'll see about this :-)*

**pass**

im\_mute()

*This is the decorator in action is printed.*

What we have is an outer function which has one parameter which is func, or function for short. Then, we have an inner function which calls func(), prints a message, and then returns the wrapper. The im\_mute() function has the decorator on top which is represented with @ and then the outer function name. A decorator is simply syntactic sugar for passing a **function inside a function**. Below:

## Magic Methods

## Meta classes

## Generators and iterator protocol

## Multiprocessing

## Networking

## C extensions

# Project #1: Your Biography

## Skills needed to complete exercise

Need to know about python strings, how to create functions, how to read user input, how to use some of the built in functions in python, and how to how an IDE or text editor. Like when creating many programs, it’s nice to have debugging and testing skills. Also, the ability to research the web and find timely answers to problems you have is also important.

Let’s create a python script to get reacquainted with ourselves. One way to do this is to ask some probing questions. Some questions that you may want to answer are things like:

* first name
* last name
* nationally
* birth place
* age
* height:
  + feet
  + inches
* weight (in pounds)
* favorite food
* favorite city

You can add on any additional questions you want. In order to create this script follow the steps:

1) In PyCharm go to the directory where you’ll place all of your programs. Right-click on the directory and select: New → Python File. Enter the name of the python file as *bio.py.*

2) In the PyCharm editor add a function named questions(). Inside the function is where all of the statements for your logic to goes inside. You can use the input() function to read in text form the terminal. If you need to read in an int or a float then you can pass the input() function into the int() or float() functions. For example, the following will read in a float from the terminal:

weight = float(input(**'Enter weight in lbs: '**))

To view a list of the builtin functions in python3 check out this url: <https://docs.python.org/3/library/functions.html>

3) Include the following code snippet after the questions() function:

**if** \_\_name\_\_ == **'\_\_main\_\_'**:

*# this is where your program starts*

questions()

This lets the python interpreter know where to start at. In every python file there’s a \_\_name\_\_ variable that’s set equal to \_\_main\_\_. Therefore, if your file explicitly includes this then it will tell the python interpreter to start here. Below is a template to how your python file will look:

def questions():

"""This is the part of the program that prompts the user"""

if \_\_name\_\_ == '\_\_main\_\_':

# this is the entry point to the program

questions()

One of the tricky things that you may want to look out for is how to read in multiple user input in a single statement. For example, reading in a single int or string is easy because you can do something like this:

>>> temperature = int(input('Temp today:'))

Temp today:75

>>> color = input('The color:')

The color:blue

However, what if you want the user to enter in two inputs so that you can store the data in feet and inches? One way to do that is to use the builtin string method called split(). What this does is split the text around a certain character like a comma.

## Sample Solution:

Below is a sample solution. Your script may have more or less questions, it really depends on how you want to create it.

def questions():

"""This is the part of the program that prompts the user

for a bunch of questions."""

first\_name = input('Enter your first name: ').capitalize()

last\_name = input('Enter your last name: ').capitalize()

nationality = input('Enter your nationality: ').capitalize()

age = int(input('Enter your age: '))

height = input('Enter feet and inches separated by commas: ')

user\_input = height.split(',')

heights = user\_input[0], user\_input[1]

weight = float(input('Enter weight in lbs: '))

favorite\_food = input('Enter your favorite food: ').capitalize()

favorite\_city = input('Enter in your favorite city: ').capitalize()

print()

print('First name: {}'.format(first\_name))

print('Last name: {}'.format(last\_name))

print('Nationality: {}'.format(nationality))

print('Age: {}'.format(age))

print('Height: {} ft {} in'.format(user\_input[0], user\_input[1]))

print('Weight: {}'.format(weight))

print('Favorite food: {}'.format(favorite\_food))

print('Favorite city: {}'.format(favorite\_city))

if \_\_name\_\_ == '\_\_main\_\_':

# this is where your program starts

questions()

Sample input:

Enter your first name: danny

Enter your last name: hill

Enter your nationality: american

Enter your age: 47

Enter feet and inches separated by commas: 5, 5

Enter weight in lbs: 200

Enter your favorite food: pizza

Enter in your favorite city: philadelphia

Sample output:

First name: Danny

Last name: Hill

Nationality: American

Age: 47

Height: 5 ft 5 in

Weight: 200.0

Favorite food: Pizza

Favorite city: Philadelphia

Note, you can run the file by opening up the terminal or commands prompt, and typing the following:

python bio.py

# Project #2: Converters and Calculators

The world is not singular. There’s many different types to something and this can vary from country to even region. Not only these tools serve as excellent learning material for exploring a new topic, but they’re also practical as it’s something that you can utilize over-and-over. There’s many converters online and Google has a massive amount that you can trigger by just entering specific queries into the search box: <https://support.google.com/websearch/answer/3284611?hl=en>

The advantage that you have as a python programmer is that you can also add new features, create updates, and write a new script for something you can’t find online. That’s the beauty about knowing how to code, you now have a new world of possibilities. In this module you’re presented with six different projects. You don’t have to do all of them (even though it’s recommended). As long as you do at least 3 of them with the only mandatory one being the temperature converter then you’ll have the green light to progress to project #3.

## Skills needed to complete these exercises:

Need to have strong researching skills, know how to use the various mathematical operators, understand conditional logic, read in user input, and know how to create modular programs by wrapping them into functions.

### Script #1: Temperature Converter

There’s three commonly used scales for measuring temperature: they’re Celsius (°C), Fahrenheit (°F), and Kelvin (K). Celsius is used by all the countries except United States, Bahamas, Belize, Cayman Islands, and Liberia. Americans that travel abroad will probably have to go through a phase in which they have to adjust to reading temperature being being measured in Celsius… I know I did! Kelvin is the unit of measurement used in the International System of Units (SI). The Kelvin scale is also heavily used in science and technology. Let’s create a script that can convert all of the temperature units to each other and back. In order to do this we must need to know the mathematical formulas. Luckily, they’re simple:

F -> C = (F - 32) x 5/9

F -> K = (F - 32)/1.8 + 273.15

C -> F = (C x 9/5) + 32

C -> K = C + 273.15

K -> F = (K - 273.15) x 9/5 + 32

K -> C = K - 273.15

With the above knowledge you can proceed to write your temperature conversion script. Sometimes the first step is the most difficult to take. Here’s some guidance if you’re struggling with how to get started.

### Create a new python file in PyCharm

As this point you should have a folder in which you store all of your python files in. In this folder create a new python file and name it: temp\_converter.py

### Think of the layout of the script

Probably one of the most important things to start thinking about is how will the script read in user input. One solution is to use python’s builtin input() function and provide the user multiple options that they can select from so that they can pick the appropriate conversion process. Here’s an example of how the script could look once the script runs:

Choose from one of the following options:

* Type ‘fc’ to convert from Fahrenheit to Celsius.
* Type ‘fk’ to convert from Fahrenheit to Kelvin.
* Type ‘cf’ to convert from Celsius to Fahrenheit.
* Type ‘ck’ to convert from Celsius to Kelvin.
* Type ‘kf’ to convert from Kelvin to Fahrenheit.
* Type ‘kc’ to convert from Kelvin to Celsius.

Then, depending on which option the user selects the appropriate function is called the calculation is done and the result is returned.

## Crafting your functions

Building off of the idea we can create the function names and just include the pass statement so that the code doesn’t do nothing yet. Here’s the name of the functions:

fahrenheit\_to\_celsius(): Converts the user input from Fahrenheit to Celsius.

fahrenheit\_to\_kelvin(): Converts the user input from Fahrenheit to Kelvin.

celsius\_to\_fahrenheit(): Converts the user input from Celsius to Fahrenheit.

celsius\_to\_kelvin(): Converts the user input from Celsius to Kelvin.

kelvin\_to\_fahrenheit(): Converts the user input from Kelvin to Fahrenheit.

kelvin\_to\_celsius(): Converts the user input from Kelvin to Celsius.

fahrenheit\_to\_celsius(): Converts the user input from Fahrenheit to Celsius.

Here’s a template of how the script looks:

**def** fahrenheit\_to\_celsius():

**pass**

**def** fahrenheit\_to\_kelvin():

**pass**

**def** celsius\_to\_fahrenheit():

**pass**

**def** celsius\_to\_kelvin():

**pass**

**def** kelvin\_to\_fahrenheit():

**pass**

**def** kelvin\_to\_celsius():

**pass**

**def** fahrenheit\_to\_celsius():

**pass**

**if** \_\_name\_\_ == **'\_\_main\_\_'**:

**pass**

## Solution

Don’t proceed to this section or less you’ve attempted to get the script to work. Not getting work is not as bad as not attempting to get it to work. Remember, you miss all of the shots you don’t take. Below is an example of how the script could be coded:

**def** fahrenheit\_to\_celsius():

temp\_in\_fahren = float(input(**'Enter the temperature in Fahrenheit '**))

celsius = (temp\_in\_fahren - 32) \* 5/9

celsius = round(celsius, 4)

print(celsius, **'°C'**)

**def** fahrenheit\_to\_kelvin():

temp\_in\_fahren = float(input(**'Enter the temperature in Kelvin '**))

kelvin = (temp\_in\_fahren - 32) / 1.8 + 273.15

print(kelvin, **'K'**)

**def** celsius\_to\_fahrenheit():

temp\_in\_celsius = float(input(**'Enter the temperature in Celsius '**))

celsius\_to\_fahren = (temp\_in\_celsius \* 9/5) + 32

print(celsius\_to\_fahren, **'°F'**)

**def** celsius\_to\_kelvin():

temp\_in\_cel = float(input(**'Enter the temperature in Celsius '**))

celsius\_to\_kel = (temp\_in\_cel + 273.15)

print(celsius\_to\_kel, **'K'**)

**def** kelvin\_to\_fahrenheit():

temp\_in\_kelvin = float(input(**'Enter the temperature in Kelvin '**))

kelvin\_to\_fahren = (temp\_in\_kelvin - 273.15) \* 9/5 + 32

kelvin\_to\_fahren = round(kelvin\_to\_fahren, 3)

print(kelvin\_to\_fahren, **'°F'**)

**def** kelvin\_to\_celsius():

temp\_in\_kelvin = float(input(**'Enter the temperature in Kelvin '**))

kelvin\_to\_cel = temp\_in\_kelvin - 273.15

kelvin\_to\_cel = round(kelvin\_to\_cel, 3)

print(kelvin\_to\_cel, **'°C'**)

**if** \_\_name\_\_ == **'\_\_main\_\_'**:

message = input(**"""Select one of the following options:**

**Type 'fc' to convert from Fahrenheit to Celsius.**

**Type 'fk' to convert from Fahrenheit to Kelvin.**

**Type 'cf' to convert from Celsius to Fahrenheit.**

**Type 'ck' to convert from Celsius to Kelvin.**

**Type 'kf' to convert from Kelvin to Fahrenheit.**

**Type 'kc' to convert from Kelvin to Celsius.**

**Enter input here:**

**"""**)

*# casefold is for case-insensitive comparisons*

message = message.casefold()

**if** message == **'fc'**:

fahrenheit\_to\_celsius()

**elif** message == **'fk'**:

fahrenheit\_to\_kelvin()

**elif** message == **'cf'**:

celsius\_to\_fahrenheit()

**elif** message == **'ck'**:

celsius\_to\_kelvin()

**elif** message == **'kf'**:

kelvin\_to\_fahrenheit()

**elif** message == **'kc'**:

kelvin\_to\_celsius()

**else**:

print(**'Not a valid option pal!'**)

You can download the script from GitHub: <https://github.com/purcellconsult/Build-Cool-Stuff-With-Python/blob/master/temp_converter.py>

## Logic Breakdown

Let’s breakdown the code into three parts: reading the user input, the conditionals, and then the main logic in the functions.

### User input:

When the program runs we shroud display a message to show the user how to use the script. We’re created a text based script with no GUI therefore instructions is a good start. To do this simply create a message in the form of a string and display it. We could of used single print statements, but I’ve opted to use a string that’s wrapped in triple quotes for this as this style of strings will make it so that we don’t have to worry about escaping characters like apostrophes. I’m referring to this segment of the code:

**if** \_\_name\_\_ == **'\_\_main\_\_'**:

message = input(**"""Select one of the following options:**

**Type 'fc' to convert from Fahrenheit to Celsius.**

**Type 'fk' to convert from Fahrenheit to Kelvin.**

**Type 'cf' to convert from Celsius to Fahrenheit.**

**Type 'ck' to convert from Celsius to Kelvin.**

**Type 'kf' to convert from Kelvin to Fahrenheit.**

**Type 'kc' to convert from Kelvin to Celsius.**

**Enter input here:**

**"""**)

The problem with reading text is that we always want to make it easy for the user to enter in text. For I.e, if the user entered FC then this will not match the condition if message == ‘fc’ and therefore the program will evaluate to False. However, with the assistance of the casefold() method all of these cases are evaluated so it doesn’t matter what the user enters.

### Conditionals

The conditionals simply check the user input, and depending on what the user enters it calls the appropriate function.

### The Functions

They prompt the user for input, and then once the data is received the appropriate calculation is done.

## Script #2: Auto loan Calculator

Cars are incredible mechanical inventions. Something that can be driven for countless of miles and still function properly is quite amazing. Like many quality things in life it costs money so in this tutorial we’ll going to code a useful script that will help us make better purchasing decisions when deciding on a new car. The script will have two parts: one, how many months it will take to pay off the loan and two, what’s the total interest paid accumulated over that period of time. Here’s what you’ll need to get started:

**Thee Formula for calculating the monthly payments on an auto loan:**

A = P x (1 + r) ^N / (1 + r)^N - 1

Formula for calculating the accrued interest:

A = P(1 + rt)

The first formula looks tricky but it’s simple once you know what all of the variables represent. Here’s a quick overview:

* P = Principal or the amount owned on a loan.
* A = Total accrued amount (principal + interest).
* r = Rate of interest per year as a decimal, or interest rate / 100.
* N = Number of months in the loan period.

If you were to go into an auto company finance department then this is the same formula that they’ll use to determine the monthly payments for your car. In the second formula, p(1 + rt), t represents the number of months on the loan, it’s just that by using r and n together (rn) would be more difficult to read.

## Script Hints

Create a function that includes the formulas for calculating each formula. For the calculating monthly payments formula you break the formula down into three parts: numerator, denominator, and then the final expression in order to reduce the errors you could get my trying to replicate the formula in a single statement. This reduces parenthesis errors and make debugging potential arithmetic errors more straightforward.

[start your coding before checking out the solution ...]

## A Solution

Below is one solution to the problem:

**def** monthly\_cost():

print(**'Gotta couple of questions for you...'**)

p = float(input(**'Enter loan amount '**))

r = float(input(**'Enter interest rate (%)'**))

n = int(input(**'Enter loan period (in months)'**))

*# convert r to a decimal and divide by interest per year*

r = (r / 100) / 12

*# breaks formula down into 3 parts to reduce error*

*# numerator*

top = r \* (1 + r)\*\*n

*# denominator*

bottom = ((1 + r)\*\*n) - 1

*# putting it all together*

a = round(p \* (top / bottom))

*# use simple interest formula*

*# I = Prt*

*# In this case, I = Prn*

total\_interest = round(p \* r \* n, 3)

print(**f'Monthly costs = ${**a**}. Total interest = ${**total\_interest**} '**)

**if** \_\_name\_\_ == **'\_\_main\_\_'**:

monthly\_cost()

The meat of this script lies in the load function of monthly\_cost. Here’s a quick breakdown of what’s happening:

1) The user input is requested in stored in the variables of p, n, and r.

2) Once the user input is collected, the mathematics is done on separate statements to minimize errors. For example, the value of r is calculated on one line, and instead of trying to translate the mathematical formula to python code on a single line, the numerator and denominator is calculated in separate statements and then combined in this statement:

a = round(p \* (top / bottom))

If you have lot’s of experience in python then doing everything in one line may be trivial. However, if you’re new to python then the important thing is to get the script to work as you can refactor (restructure) the code later.

The auto loan calculator script on GitHub: <https://github.com/purcellconsult/Build-Cool-Stuff-With-Python/blob/master/auto_loan_calculator.py>

## Script # 3: Mortgage Calculator

Getting a house is the American dream, but to obtain a dream does costs. In this exercise we’re going to create a script that calculates the monthly payment that someone owes on a house. When one takes out a loan for a house this is known as a mortgage. Mortgage calculators can get pretty complex, so in this exercise we’re going to *stupify* the process so that we just calculate the monthly payments and total mortgage that one would owe contingent on some variables. These variables are:

* **Mortgage period**: How long the mortgage will last in years.
* **Principal**: The amount of money owed on the loan.
* **Interest rate**: The percentage of principal charged by the lender for the use of it’s money.

After digging around here’s the formula that you can use to calculate the monthly payments on a house:

* p x r (1 + r) ^ N / (1 + r) ^ N - 1

Put together a python script that prompts the user for their name, principal, interest rate, and then outputs the monthly payment and the total interest a homeowner would pay.

[start your coding before checking out the solution ...]

## A Solution:

**def** mortgage():

name = input(**'Enter your name '**).capitalize()

print(**f"Time to calculate your mortgage payments {**name**}... "**)

principal = float(input(**'Enter in your principal '**))

interest\_rate = float(input(**'Enter interest rate '**))

r = (interest\_rate / 100) / 12

n = int(input(**'Enter mortgage period (years) '**))

*# get total number of months*

n = n \* 12

numerator = r \* (1 + r) \*\* n

deno = (1 + r) \*\* n - 1

monthly\_payment = principal \* (numerator / deno)

monthly\_payment = round(monthly\_payment)

total\_mortgage = monthly\_payment \* 30 \* 12

print(**f'Monthly payment: ${**monthly\_payment**}, total mortgage = ${**total\_mortgage**}'**)

**if** \_\_name\_\_ == **'\_\_main\_\_'**:

mortgage()

This script is very similar to the auto loan script except that the formula varies:

* The variables of principal, interest rate, r, and n are prompted from the user.
* The numerator and denominator of the function is calculated separately and then combined together at the end to emulate the formula: p x r (1 + r) ^ N / (1 + r) ^ N – 1
* Once the total monthly payment is calculated, the total mortgage is computed by taking monthly payment and multiply it by 30 and 12. The reason for this is because there’s roughly 20 days in a month, and there’s 12 months in a year.

## Script # 4: [Sports Betting Calculator](https://github.com/purcellconsult/Build-Cool-Stuff-With-Python/blob/master/sports_betting.py)

## Las Vegas, aka Sin City is a city that’ quite unlike any other. A now desert oasis that was once a front to a shady underworld. Not only can you get married at a wedding chapel, feast on cuisines by world class chefs, and party at one of the pricey and crowded super clubs, you can also lose all of your money as well. We’re going to explore one of the features in Vegas that attracts visitors worldwide which is the gaming, and more explicitly, sports betting. We’re going to write us a little python script that allows us to calculate American, Fractional (European), and Decimal odds along with Implied Probability. Below is an explanation of the various terms and how to compute them:

## **American:** Also known as money line odds or US odds. The odds are preceded by either a plus or minus symbol. If the odds includes a minus, then that indicates the favorite and represents the amount you need to win $100. If it includes a + then that indicates the underdog and the amount won for ever $100 staked. A bet or wager is the amount of money you wish to risk.

## **Fractional**: Also known as UK odds are typically used in the UK and Ireland. The numerator represents the amount your wager will yield and the denominator represents the amount that’s bet. For example, 20/15 means you bet $150 to win $200.

## **Decimal**:Also known as European or continental odds, is a form of sports betting that’s popular in Continental Europe, Australia, New Zealand, and Canada. The decimal odds represent the amount that a better wins for every $1 wagered. What makes decimal odds different from American and Fractional odds is that your stake is already included in the decimal number, therefore no need to add back your stake. Therefore, the formula for computing decimal odds is:

## *Total Return = Stake x Decimal Odd Number*

## If you have calculated the fractional odds, then you can translate the fractional odds to decimal odds simply by adding a 1 to it.

## **Implied Portability**: This is the conversion of betting odds into a percentage and that’s important to know if you want to become a professional sports better because it helps you access the potential value on a particular market. You can calculate the implied probability for American, fractional, and decimal odds by using the following formulas:

### American:

## The equation to convert negative American odds:

## Negative American odds / (Negative American odds + 100) \* 100 = implied probability

## The equation to convert positive American odds:

## 100 / (positive American odds + 100) \* 100 = implied probability

### Fractional odds:

## The formula to calculate implied probability from fractional odds:

## denominator / (denominator + numerator) \* 100 = implied probability

### Decimal odds:

## The formula for calculating implied probability from decimal odds is:

## (1/ decimal odds) \* 100 = implied probability

## Script hints

Write a python script that prompts the user to enter the American odds for a sporting event, and then the script computes the respective fractional/decimal odds and implied probability. To make it easier to compute the fraction odds consider using the fractions module in python: <https://docs.python.org/3/library/fractions.html>

[start your coding before checking out the solution …]

## A solution:

**from** fractions **import** Fraction

**def** odds\_calculator(american\_odds, amount=100):

*"""provides the amount to win and the payout."""*

**if** american\_odds > 0:

fractional\_odds = Fraction(american\_odds, 100)

to\_win = float(fractional\_odds \* amount)

payout = float(to\_win + amount)

decimal\_odds = 1 + fractional\_odds

decimal\_odds = float(decimal\_odds)

implied\_prob = round(100 / (american\_odds + 100), 3)

implied\_prob \*= 100

print(**'-----------------------'**)

print(

**f'To win: {**to\_win**} \n'**

**f'Payout: {**payout**} \n'**

**f'American odds: {**american\_odds**} \n'**

**f'Fractional odds: {**fractional\_odds**} \n'**

**f'Decimal odds: {**decimal\_odds**} \n'**

**f'Implied probability: {**implied\_prob**} '**

)

**else**:

american\_odds = (american\_odds)

fractional\_odds = abs(Fraction(100, american\_odds))

to\_win = int(amount \* fractional\_odds)

payout = int(to\_win + amount)

decimal\_odds = 1 + fractional\_odds

implied\_prob = (-1 \* american\_odds)

implied\_prob = round(implied\_prob / ((-1 \* american\_odds) + 100),3)

implied\_prob \*= 100

print(

**f'To win: {**to\_win**} \n'**

**f'Payout: {**payout**} \n'**

**f'American odds: {**american\_odds**} \n'**

**f'Fractional odds: {**fractional\_odds**} \n'**

**f'Decimal odds: {**decimal\_odds**} \n'**

**f'Implied probability: {**implied\_prob**} '**

)

**if** \_\_name\_\_ == **'\_\_main\_\_'**:

print(**'Welcome to the odds calculator: '**)

odds = int(input(**'Enter the odds '**))

wager = int(input(**'Enter wager (bet amount)'**))

print(**f'Bet {**wager**}'**)

odds\_calculator(odds, wager)

## Solution Explanation:

The bulk of the action takes place in odds\_calculator(). The function accepts two arguments which are american\_odds and wager. These variables are gathered when the script starts executing. From there, the function is separated into two branches. One, when american\_odds is greater than 0, and the other when it’s less than 0. Remember, American odds either includes a negative or plus sign in front of the integer, so contingent on the user input two separate branches needs to be created to allow for the appropriate calculations.

Download the sports betting calculator script:<https://github.com/purcellconsult/Build-Cool-Stuff-With-Python/blob/master/sports_betting.py>

## Script # 5: Top 10 World's Global Economy Currency Exchange Calculator

Dinero makes the world goes world, and in this project we’re going to prove it. We’re going to write a currency converter which allows us to convert the world’s top 10 global economies to one another. Before we can do this we need to have a couple of things figured out: what are the top 10 global economies, and what is the proper currency exchange rate between them. The problems with this is that these two variables are contingent to change in the future but in the meantime here’s the information that’s available via some web based resources:

1) Top Economics: <https://www.focus-economics.com/blog/the-largest-economies-in-the-world>

2) XE Currency Converter: <https://www.xe.com/currencyconverter> (converts from one currency to another)

## Script hints:

The logic to this script is pretty straightforward. Once you write one function that converts one country currency to another, the logic is essentially the same but the only thing that changes are the currency exchange rates. To give you a hint on how to get started here’s one way to approach the problem:

**if** \_\_name\_\_ == **'\_\_main\_\_'**:

print(**'Enter the currency you have. Provide the alphabetic code.\n'**

**'I.e: USD for Dollar or JPY for yen.\n'**)

countries = input(**'Need to see a list of countries available? Enter "y"'**

**'or "n" '**).lower()

**if** countries == **'y'**:

available\_currencies()

**else**:

print(**'That\'s fine :-)\n'**)

code = input(**'Enter the alphabetic code: '**).upper()

**if** code == **'USD'**:

america()

**elif** code == **'CNY'**:

china()

**elif** code == **'JPY'**:

japan()

**elif** code == **'INR'**:

india()

**elif** code == **'CAD'**:

canada()

**elif** code == **'GBP'**:

the\_uk()

**elif** code == **'BRL'**:

brazil()

**elif** code == **'EU' or** code == **'BRD' or** code == **'ITL' or** \

code == **'FR'**:

euros()

**else**:

print(**'Incorrect input. Enter a correct one'**)

The above code snippet shows the first portion of the program that gets executed. Since this is purely a text based script then providing the user instructions on how to use it is a good idea. Therefore, there’s a conditional that asks the user if they want to see a list of countries that’s available in the script. If they choose yes then the data is printed, if not then the control flows into the next portion of the script. The user input is requested for a country code to convert from. Depending on the option that the user enters a function that matches the country name is executed, and if none of the correct input is entered then the program prints a message and terminates. Below is an idea of how to approach building the functions for the respective countries:

**def** america():

*"""Converts from American dollar to various currencies """*

print(**'Convert from USD to some other currency: '**)

symbol = **None**

dollars = input(**'Enter the currency to convert to: '**).upper()

usd\_amount = float(input(**'Enter the amount of dollars in USD: '**))

**if** dollars == **'CNY'**:

symbol = **'¥'**

conversion = usd\_amount \* 6.73

**elif** dollars == **'JPY'**:

symbol = **'¥'**

conversion = usd\_amount \* 111.11

**elif** dollars == **'EU'**:

symbol = **'**€**'**

conversion = usd\_amount \* .89

**elif** dollars == **'GBP'**:

symbol = **'£'**

conversion = usd\_amount \* .76

**elif** dollars == **'INR'**:

symbol = **'**₹**'**

conversion = usd\_amount \* 69.18

**elif** dollars == **'BRL'**:

symbol = **'R$'**

conversion = usd\_amount \* 3.94

**elif** dollars == **'CAD'**:

symbol = **'$'**

conversion = usd\_amount \* 1.34

**else**:

**return**

print(**f'{**usd\_amount**}** **USD to {**dollars**}** **is {**conversion**} {**symbol**}'**)

Here’s a breakdown of the components of this function:

* The country code to convert the currency to is prompted.
* Depending on the output that the user selects, the program creates the appropriate currency symbol, and then does the conversion by storing the output in the conversion variable. If none of the valid country codes are entered then the program imply returns.
* The output is printed.

Once you have the logic for one country’s function figured out, then it’s more or less the same logic except that you use different numbers for exchange rates.

[start your coding before checking out the solution ...]

## A solution:

The full solution contains code that would span several pages and therefore be difficult to read in an e-book or paperback format. Therefore, it’s suggested to download the script from GitHub to look over it: <https://github.com/purcellconsult/Build-Cool-Stuff-With-Python/blob/master/currency_converter.py>

## Script #6: Spanish translator

Language translation is a complicated art. It’s not something that’s easily done by using a translation tool online because there’s many elements that will be missing and will typically make translated sentences look like mumbo-jumbo. Therefore, to serve as a learning tool we’re going to simplify the process and focus on a narrow domain of words and phases to transliterate. Not only is this a good exercise to gain more familiarity with python data structures, strings, and conditional statements, it’s also a good exercise for gaining more comfort with loops in python as depending on what route you choose to solve the problem you can use them several times. So, to simplify the process create a python script that focuses on translating foods and general phases. Therefore, here’s how the outline of the script I created looks:

def food():

pass

def general\_phases():

pass

if \_\_name\_\_ == '\_\_main\_\_':

print('Bienvenidos! What phases would you like to translate?')

print('1: Common foods ')

print('2: General phases')

your\_choice = int(input("Enter your choice: '1' or '2'"))

if your\_choice == 1:

food()

elif your\_choice == 2:

general\_phases()

else:

print('Not a possible choice!')

The list and food items and general phases that you want to translate is up to you. I would suggest using a dictionary to store the English and Spanish mappings, as this data structure is perfectly suited for this type of task.

[start coding…]

## A Possible Solution

Spanish translator app: <https://github.com/purcellconsult/Build-Cool-Stuff-With-Python/blob/master/spanish_translator.py>

## Module III Recap

In this module we learned how to start putting together our python programming skills to start building some real-world practical programs. Converters and calculators are some basic programs that you can start writing that allows you to start solving real world problems. If you built a couple of these scripts then this is something that you can show to to your friends and family to showcase your evolving python programming skill set.