# 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: Guessing Game

Project #4: Rock Paper Scissors

Project #5:

Project #6:

Project #7:

Project #8:

Project #9:

Project # 10:

# 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

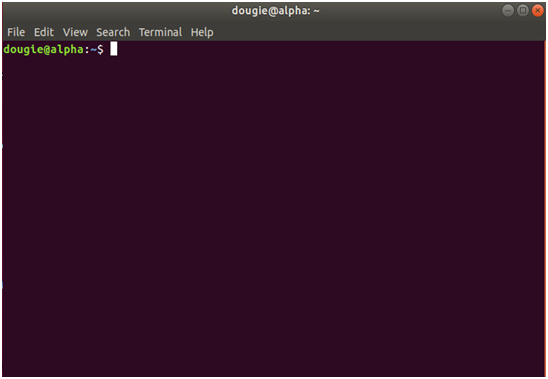
If the output shows that Python 3 is installed then you’re safe… for now. If you get an error then that’s not cool and you have some work to do. You can fix this by downloading and installing Python with the [appropriate Mac OS X installer](https://www.python.org/downloads) that matches your system.

### **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

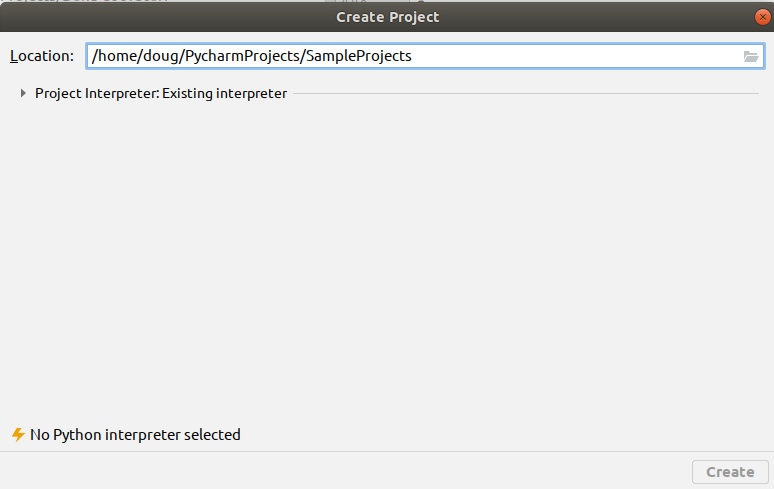
## Hello World With PyCharm

# 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.

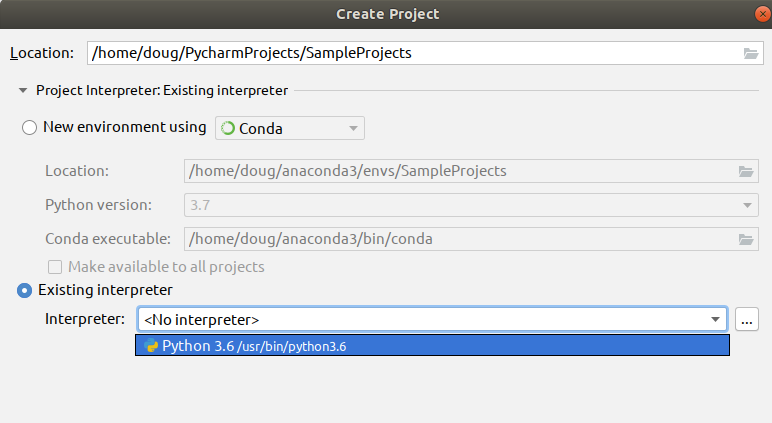
### Create a Fresh Project

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

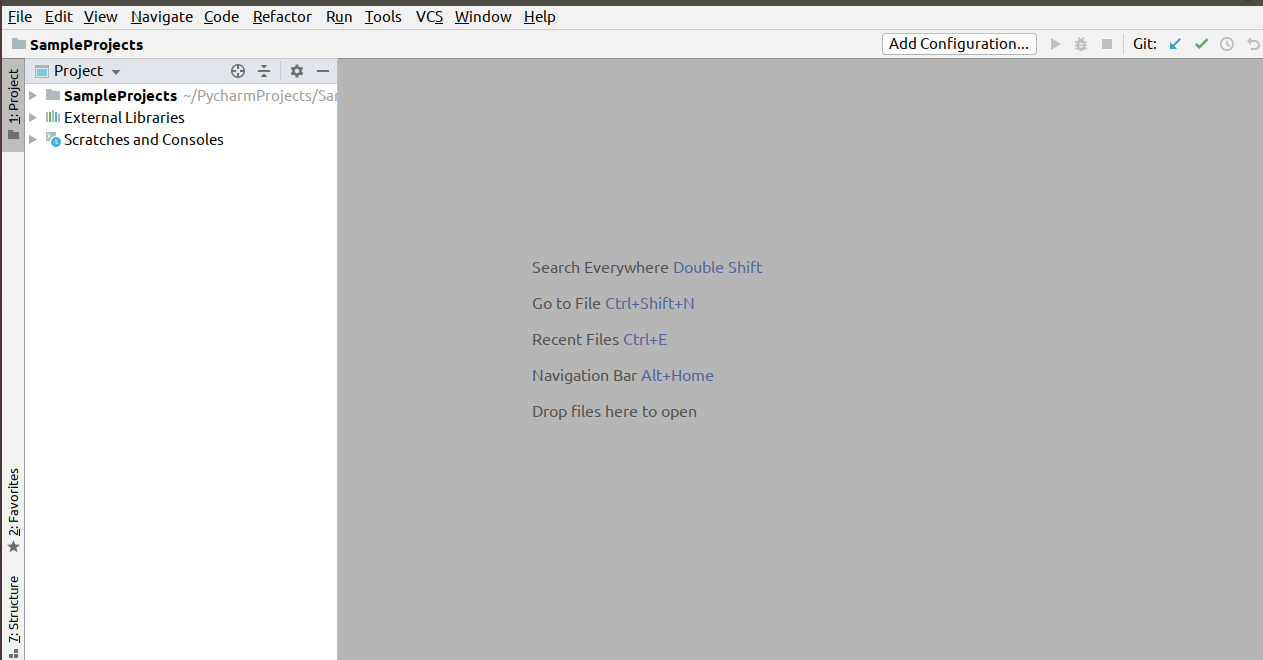
Name the project *SampleProjects*. A project is an organizational unit in PyCharm. Here’s a screenshot of what the setup should look like thus far:



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:



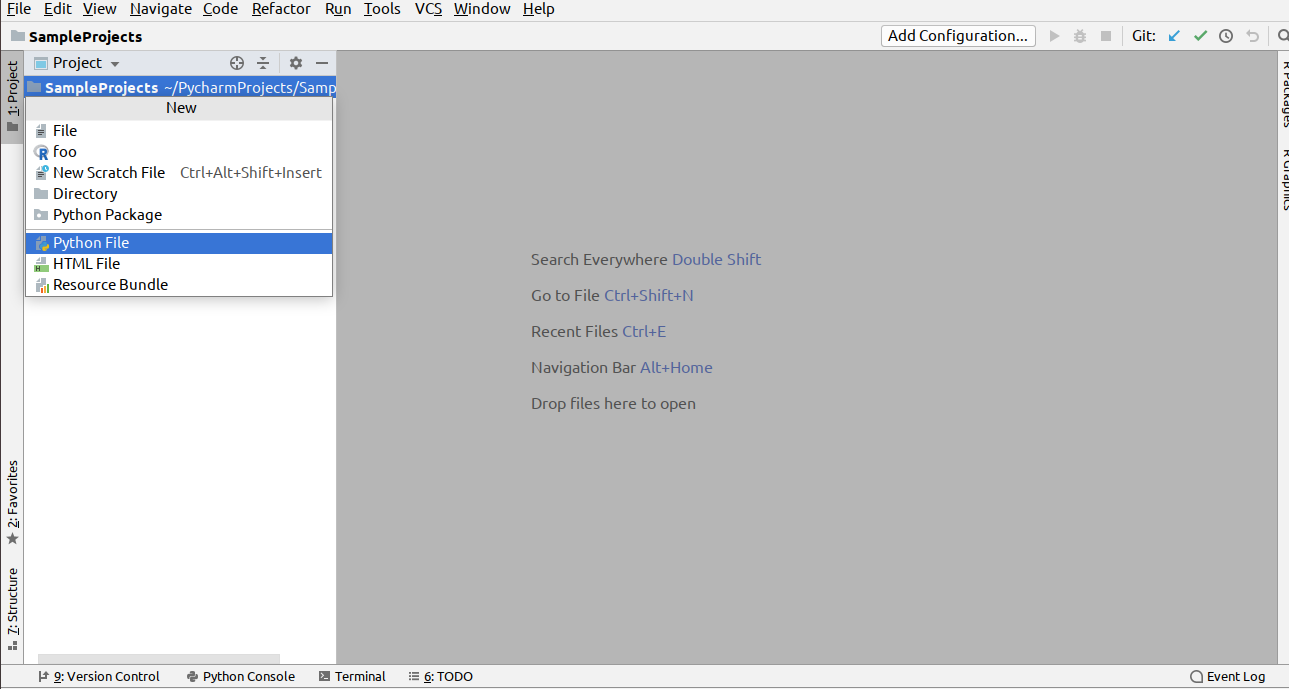
Once the python interpreter is selected click the create button to create your project. Here’s a screenshot of the setup:



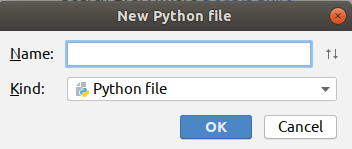
### Create a fresh python file

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

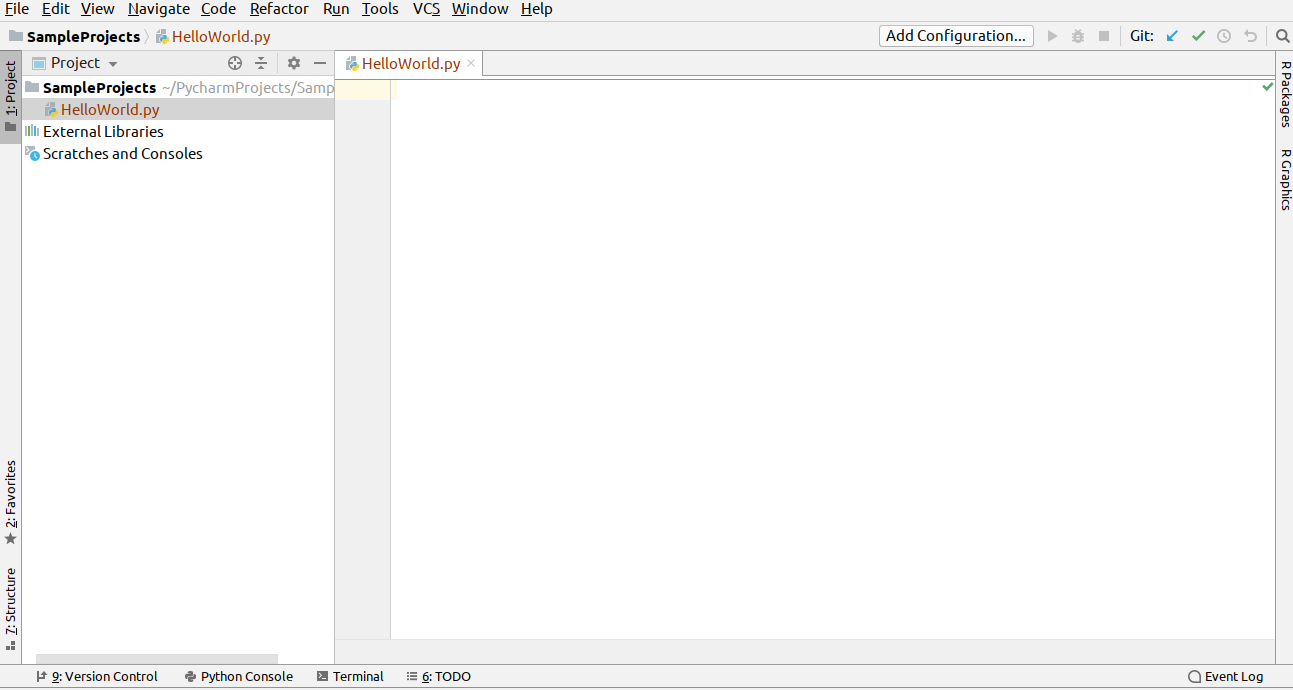
**File → New File → python file.**



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.

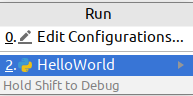
### Add code and run the file

Copy the following code into the editor:

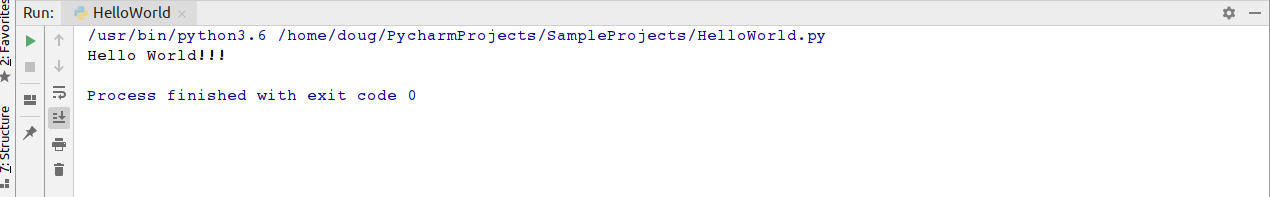
print(**'Hello World!!!'**)

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.

# 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))

## 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)

## 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())

# Data structures in python3.6

There are four builtin data strictures 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()

# adds an object to the list

evens.append(100)

# merges another list with the list

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

# pops an item from the list

evens.pop()

# iterating over a list

for x in evens:

print(x)

## 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)

## 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)

## Sets

Stores unique items.

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

print(sorted(letters))

## 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)

## Output:

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()

**Here’s the output:**

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

## 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 inner 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