Introduction to Programming Fundamentals & Python

REFERENCE BOOK

learnr.

Welcome to the Reference Book 4
Resources Location 5
Python Data Types 6
Setting the Data Type of a Variable 7
Setting the Type Using Constructor Methods 8
How to Find out the Data Type of a Variable Programmatically 9
Why Do We Need to Find Out the Type? 9
Lists: A Deeper Understanding 10
Python For-Loops and IN Keyword 11
Algorithm Efficiency & Complexity Classes 12
Introduction 12
How to Determine the "Cost of an Algorithm" – Theory 13
How to Determine the "Cost of an Algorithm" – Example 1 14
How to Determine the "Cost of an Algorithm" – Example 2 15
Efficiency of Algorithms, Complexity Class/Big-O Notation – A Deeper Look 15
Setting up a Full Development Environment 16
Introduction 16
What are the Components of an Environment? 16
Step 1 – Downloading Anaconda (Our Interpreter) 17
Step 2 – Installing Anaconda (Our Interpreter) 18
Step 3 – Downloading PyCharm (Our IDE) 19
Step 4 – Installing PyCharm (Our IDE) 20
Step 5 – Creating a Project 21
Step 6 – Importing Packages (NumPy) [OPTIONAL] 25
Reading Data from a .CSV File 29
Introduction 29
Step 1 – Import the "CSV" Package 30
Step 2 – "Opening" the .CSV File 30
Step 3 – Creating the Iterator 31
Step 4 – Extracting & Printing the Data (Each Row As List) 32
Step 5 – Extracting & Printing the Data (Each Value in File on Separate Line) 33
Writing Data to a .CSV File 34
Step 1 – Import the "CSV" Package 34
Step 2 – "Opening" the .CSV File 35

Step 3 – Creating the Writer 36
Step 4 – Creating the Writer 36

Welcome to the Reference Book

First and foremost, thank you for your interest in learning programming. This reference book is designed to support your learning during the live sessions and supplement your learning after the sessions.

Unlike other learning resources this reference book is written in a casual conversational manner while using simple and easy to understand vocabulary (where possible).

We believe when attempting to convey complex subject matter which is hard enough on its own to understand there is no value added by using complex sentence structure and vocabulary.

We also don't believe in reinventing the wheel when creating our content, so if there is a resource we find on the internet which can further your learning, we point you in that direction. Where some content is lacking in depth or understandability then we either create our own material for that or combine it with information found online.

Resources Location

Please find the questions, solutions and files here:
https://github.com/muke101/learnr-solutions

Online Compiler/IDE used in lectures:
https://repl.it/languages/python3

Python Data Types

As discussed in lectures data types are an important concept in programming. Python is a dynamically typed language[1]. During the lectures you were introduced to a some of the data types (string, float, integer) – we will go into some more depth here.

Python has the following data types built-in by default, in these categories:
Text types: str
Numeric Types: int, float, complex
Sequence Types: list, tuple, range
Mapping Types: dict
Set Types: set, frozenset
Boolean Type: bool
Binary Types: bytes, bytearray, memoryview

A factor that must be noted is while dynamically typed languages can infer the type of the variable by whatever value is assigned to the variable at that point in time the interpreter can sometime get this incorrect as programming complexity increases. Therefore, there will be cases where you may wish to formally specify the type at declaration.

[1] A dynamically typed language is a language which doesn't require the programmer to specify the "type"/"data-type" of a variable during declaration. **Find out more:** https://stackoverflow.com/questions/1517582/what-is-the-difference-between-statically-typed-and-dynamically-typed-languages

Setting the Data Type of a Variable

As aforementioned as Python is dynamically typed, we don't have to formally "set" or "specify" the data type (however, for more complex programs it may be valuable). Instead, in Python simply assigning a value to a variable sets the "type" of the variable (due to the dynamic typing in Python).

Therefore, whatever a variable holds at a point in time is the type of the variable. So technically in Python setting the data type is automatically done during declaration and then by future assignment the type may change. See below for a list of variables with all the data types.

Example	Data Type
`x = "Hello World"`	str
`x = 20`	int
`x = 20.5`	float
`x = 1j`	complex
`x = ["apple", "banana", "cherry"]`	list
`x = ("apple", "banana", "cherry")`	tuple
`x = range(6)`	range
`x = {"name" : "John", "age" : 36}`	dict
`x = {"apple", "banana", "cherry"}`	set
`x = frozenset({"apple", "banana", "cherry"})`	frozenset
`x = True`	bool
`x = b"Hello"`	bytes
`x = bytearray(5)`	bytearray
`x = memoryview(bytes(5))`	memoryview

Setting the Type Using Constructor Methods

As complexity of your program increases there may be scenarios where it is worthwhile using Python's Constructor Methods to specify the data type of a variable to prevent any issues with incorrect inference of data types at run time.

We can do this during variable declaration. See below for a list of variable declarations for all the Python Data Types with the types specified using the Constructor Methods.

Example	Data Type
`x = str("Hello World")`	str
`x = int(20)`	int
`x = float(20.5)`	float
`x = complex(1j)`	complex
`x = list(("apple", "banana", "cherry"))`	list
`x = tuple(("apple", "banana", "cherry"))`	tuple
`x = range(6)`	range
`x = dict(name="John", age=36)`	dict
`x = set(("apple", "banana", "cherry"))`	set
`x = frozenset(("apple", "banana", "cherry"))`	frozenset
`x = bool(5)`	bool
`x = bytes(5)`	bytes
`x = bytearray(5)`	bytearray
`x = memoryview(bytes(5))`	memoryview

How to Find out the Data Type of a Variable Programmatically

A human can read code and determine what the "type" of a variable is. But, how do we find out via code/programmatically what data type a variable is?

We use **type().**

Example

```
x = 10 //This line declares a variable and assigns "10" to x
print(type(x)) //This line uses the function "type()" and in the parentheses we pass the
                variable x as an argument.
```

Console Output from code above: <class 'int'>

Why Do We Need to Find Out the Type?

This one is very simple. Depending on what you are attempting to create you may want to make the code behave in different ways depending on the incoming data.

For example, if variable is a string do X, if the same variable is an integer do Y.

Lists: A Deeper Understanding

As mentioned in lectures there are a lot of operations that can be achieved with lists. We found a very good resource which explains everything to do with lists.

All credits go to the great minds at Programiz:
https://www.programiz.com/python-programming/list

Python For-Loops and IN Keyword

In lectures we covered the concepts of for loops, they allow us to execute a block of code a fixed number of times.

In lectures you were shown for-loops using the range() function, where you had to specify the start value of the loop and the end value of the loop. For example, the loop counter starts are 0 and finishes at 100 where the difference between the two values is the number of times the nested code block will execute.

In Python (and other languages) there is usually an "enhanced" for-loop or something similar which provides the functionality of a normal for-loop but instead is more suited for data extraction.

Take a list of values

```
cryptoCoinList = ["BitCoin","Ethereum", "Ripple", "Dogecoin"]
```

We can use the standard for-loop with the range function index into the list and print those values like this:

```
cryptoCoinList = ["BitCoin","Ethereum", "Ripple", "Dogecoin"]
for i in range(0,3):
  print(cryptoCoinList [i])
```

However, in Python we can use the "IN" keyword combined with for-loops and iterable data types like lists. As shown below:

```
cryptoCoinList = ["BitCoin","Ethereum", "Ripple", "Dogecoin"]
for coinName in cryptoCoinList:
  print(coinName)
```

The output is the same, however no indexing is necessary with the "in" keyword.

Algorithm Efficiency & Complexity Classes

Introduction

Why do we have to bother with efficiency in our algorithms when computers can perform an unimaginable number of calculations in incomprehensible slices of time?

…Well you aren't wrong BUT as we increase the scale of our program and the size of our dataset, inefficient algorithms quickly become a very big issue.

For example, the domain of high frequency trading[2] requires processing and analysing exceedingly large datasets and then executing a trade (buy or sell x) as fast and accurately as possible.

In our example we will say we have two algorithms where the task is:
"Buy 50 shares of a stock when its 50-day moving average goes above the 200-day moving average."

The dataset the two algorithms are based on is the same but the only factor which differs is the implementation of the algorithm to execute the task. The efficient algorithm will be able to analyse all the data points and execute the trade faster than the inefficient one – therefore gaining an advantage over the other algorithm as the price per share may have changed in the time elapse.

As we know the time to enter the market or exit is crucial and can have a big impact on profit margins as well as minimising losses – therefore efficiency is vital.

In more human terms, the faster a human can process all the information provided accurately to reach decision based on the data the better they will perform.

---

[2] To find out more about high frequency trading visit:
https://www.investopedia.com/articles/active-trading/101014/basics-algorithmic-trading-concepts-and-examples.asp

https://www.ft.com/video/6e7cd4df-18c2-440a-8b20-01ed789d556d

How to Determine the "Cost of an Algorithm" – Theory

The cost of running an algorithm is typically **based on the size of the input.**

The efficiency of the algorithm is expressed as a cost function on the size of input (don't worry too much right now we will explain this further).

…But how do we actually determine the cost then?

We have to ensure these factors are discounted when measuring the cost:

1. Should not be influenced by speed of computer
2. Should not be influenced by choice of programming language
3. Should not be influenced by ability of programmer

Instead we focus on counting the number of significant actions in the algorithm.

What are Significant Actions?

Significant actions depend on the domain of the problem you are attempting to address but most commonly we can say the following actions are significant for these domains:

1. **Sorting** – Count the number of comparisons between items
2. **Searching** – Count the number of comparisons between items
3. **Summing** – Count the number of arithmetic operations

In majority of scenarios algorithms at their root do one or a combination of the above (sorting/searching/summing) in order to solve a problem.

How to Determine the "Cost of an Algorithm" – Example 1

Algorithm Name: Summing numbers
Legal Inputs: Positive integer *n*
Required Outputs: Sum of first *n* positive numbers

```
positiveNum = 3 //(our n)
SUM = 0
ITER = 1
while ITER <= positiveNum:
  SUM = SUM + ITER
  ITER = ITER + 1
print(SUM)
```

OUTPUT: 6

Explanation
The while loop is performing a comparison when the loop variable "ITER" is compared against the variable "positiveNum" which has a value of 3.

As we can see the loop variable "ITER" is incremented by one each time. Therefore, the loop will execute three times i.e. three comparisons will be made. If we look closely the number of loop executions are dictated by the value of the variable "positiveNum" i.e. our "n"-size dataset/input.

Finally, within our while-loop we perform two arithmetic operations. Each time the loop executes so do the arithmetic operations – therefore we would multiply by the number of times our loop runs i.e. "n".

For illustrative purposes let's use real numbers. The loop executes 3 times and each time so do the arithmetic operations.

The total number of significant actions for this algorithm is 6 (with this data input of 3).

Employing algebra we know that the amount of significant actions are dictated by the value held by the variable "positiveNum". For example if the number was 4 then the number of significant actions in this algorithm would become 8.

Therefore, let 3 become "n". Now we can say the number of significant actions will be 2n. Two significant operations (arithmetic) from the contents of the while loop multiplied by "n".

However, to express the in general format we only include the highest cost which would be "n". **This is our "complexity class"/Big O Notation.**

FINAL ANSWER: The cost of running this algorithm is *n*. The larger "n" i.e. the dataset is, the less efficient the algorithm is. We can express the complexity class of this algorithm as "linear".

How to Determine the "Cost of an Algorithm" – Example 2

Algorithm Name: Summing numbers
Legal Inputs: Positive integer *n*
Required Outputs: Sum of first *n* positive numbers

```
positiveNum = 3
print(positiveNum*(positiveNum + 1)/2)
```

OUTPUT: 6

Explanation
The algorithm above achieves the exact same output, however, only has three significant actions.

- One action to add the value held by "positiveNum" to 1
- One action to divide the result of the previous action by 2
- One action to multiply the result of the previous action by the value held by "positiveNum"

The difference with this equation is that regardless of what value is held by the variable "positiveNum" there will only EVER be 3 significant actions occurring.

FINAL ANSWER: The cost of running this algorithm is *constant*. Regardless of the size of the input/dataset the efficiency is unaffected. We can express the complexity class of this algorithm as "constant". This is the most efficient type of algorithm.

PLEASE NOTE: While efficiency is important, for your stage of development a good rule of thumb is the length of the code and ask yourself is this the MOST straight forward way of doing this task? Lastly, some solutions to problems can only be so efficient. Often when problems are very complex "highly efficient" algorithms aren't possible and you will have to settle for something "less efficient" but that is just the nature of some problem – so don't worry!

Efficiency of Algorithms, Complexity Class/Big-O Notation – A Deeper Understanding

As with many things YouTube often explains the trickiest concepts best.

All credits go to the Colt Steele, an incredible teacher of all things Computer Science:
https://www.youtube.com/watch?v=kS_gr2_-ws8

Setting up a Full Development Environment

Introduction

This guide will walk you through setting up a fully featured and functioning Python environment on your own computer. This is especially useful if you want to do work with particular Python packages or work with files downloaded on your computer.

WHAT'S AN ENVIRONMENT? An 'environment' simply refers to the sequence programs you use to develop a program and how they're configured in your operating system (in your case, windows).

What are the Components of an Environment?

There are usually two main components to a python environment:

1. The interpreter
2. The IDE/text editor

The interpreter is simply Python itself, it's the thing that takes your python code and executes it on your computer.

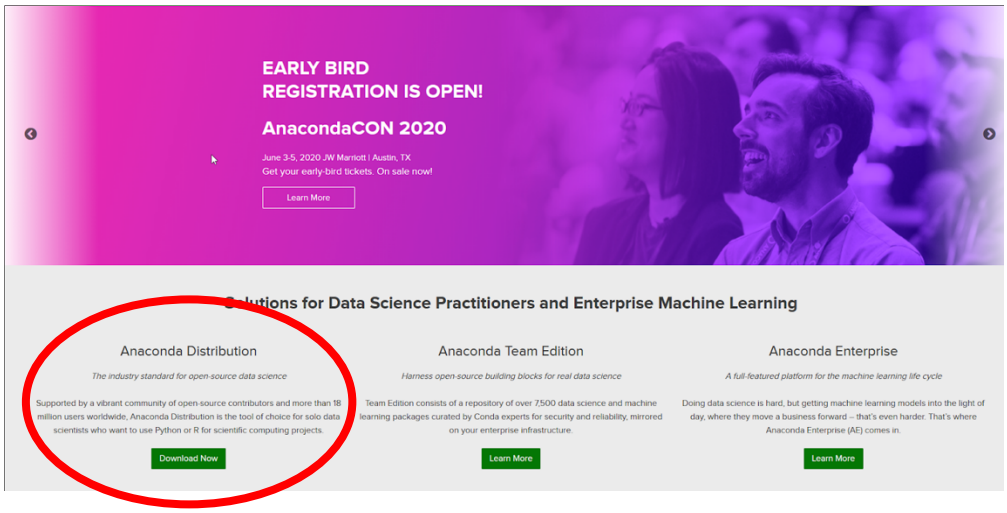
Sitting one layer above is we need something to write code with easily, and for this we can either use a regular text editor (even notepad is an example) or an 'Integrated Development Environment', or 'IDE'. The latter is similar to a text editor but with many more developer focused. An IDE makes programming easier (especially on larger projects) and allows one to create a full scale application.

For the purposes of this walkthrough, we'll be showing you how to get set up using the IDE 'PyCharm'.

We'll also be using a special version of our Python interpreter called 'Anaconda'. This is like regular Python but it comes with some special configurations and packages pre-installed to allow us to jump into developing complex programs much faster.

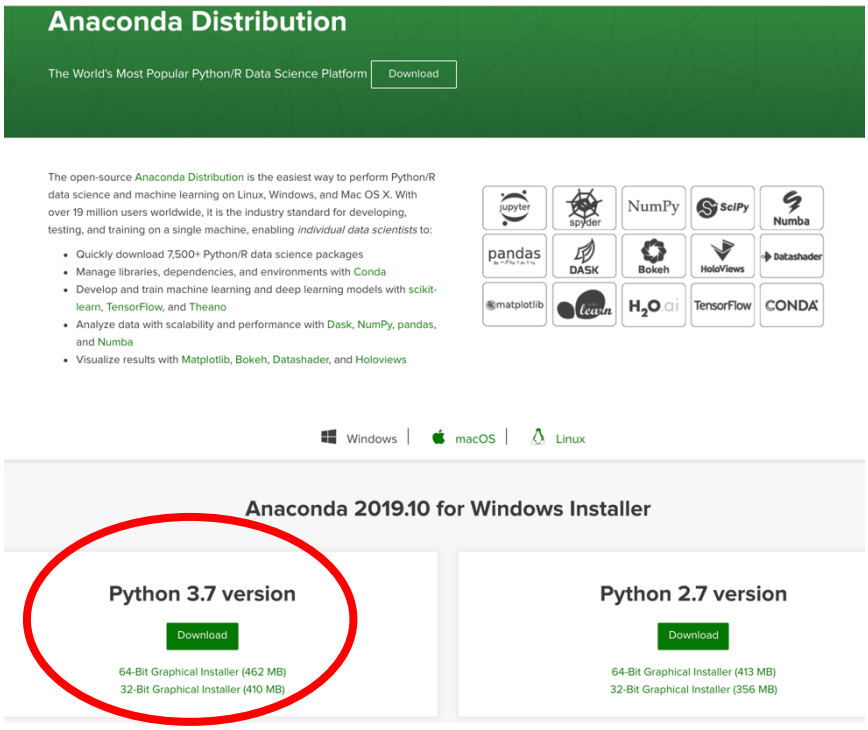
Step 1 – Downloading Anaconda (Our Interpreter)

First off, let's download and install Anaconda. Go to the anaconda website at www.anaconda.com and click 'Download Now' (circled in red).



You should then be presented with a screen similar to the one you see below. Under 'Python 3.7 version' click '64-bit graphical installer' (circled in red).

PLEASE NOTE: Select 64-bit if your Windows machine is 64-bit, otherwise select 32-bit. To find out whether your machine is 32-bit or 64-bit please follow this link: https://www.computerhope.com/issues/ch001121.htm

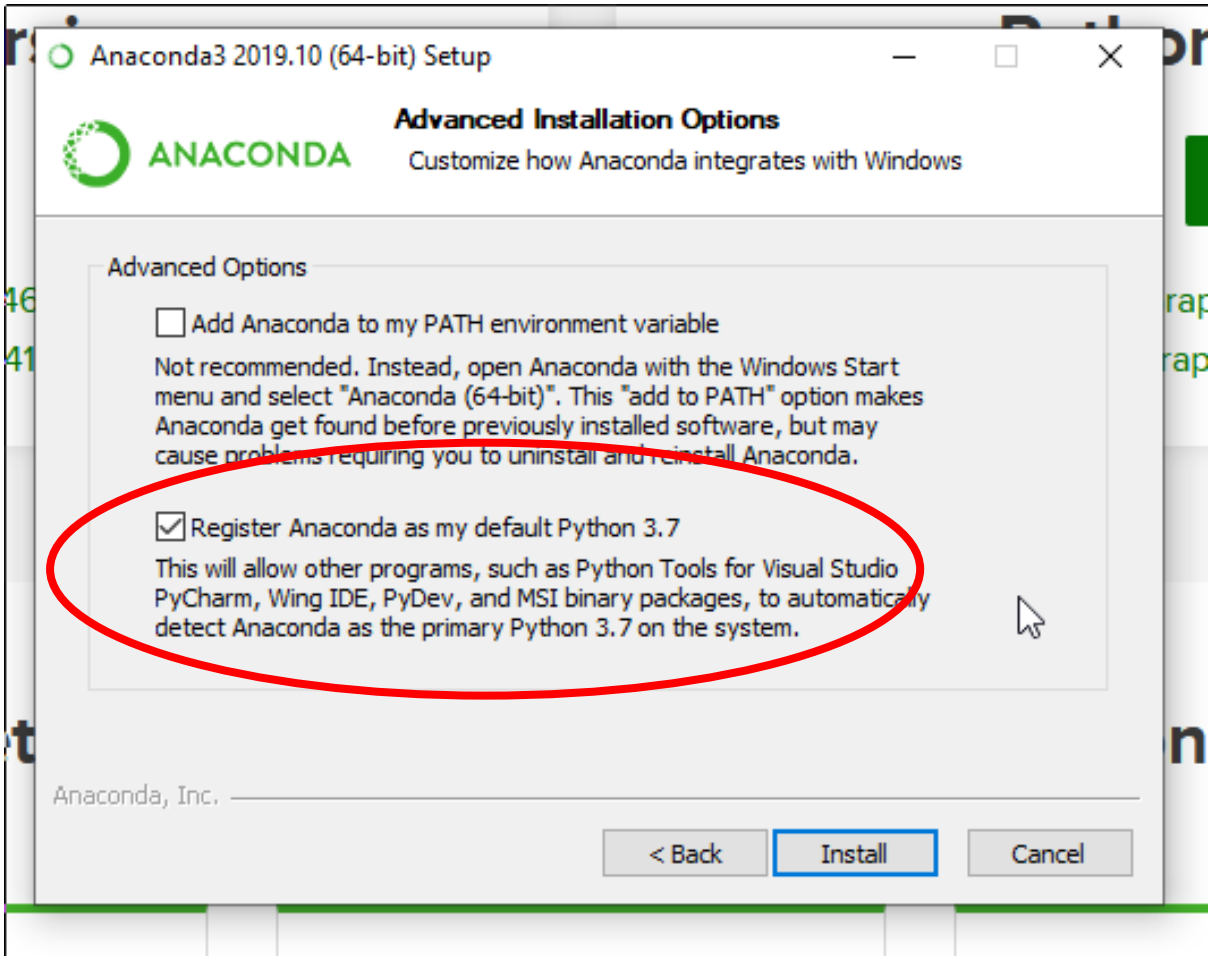


Step 2 – Installing Anaconda (Our Interpreter)

Once downloaded, find the file and run it. An installation wizard should open and follow the instructions provided.

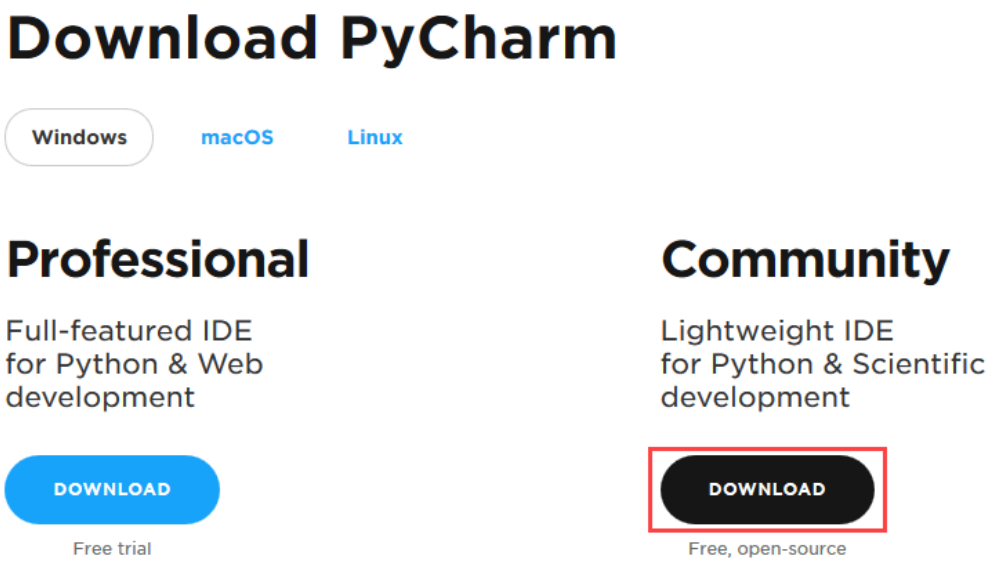
When asked, be sure to tick 'Register Anaconda as my default Python 3.7', and only this (circled in red).

Anaconda should now be installed. Next we'll need to install PyCharm (our IDE).



Step 3 – Downloading PyCharm (Our IDE)

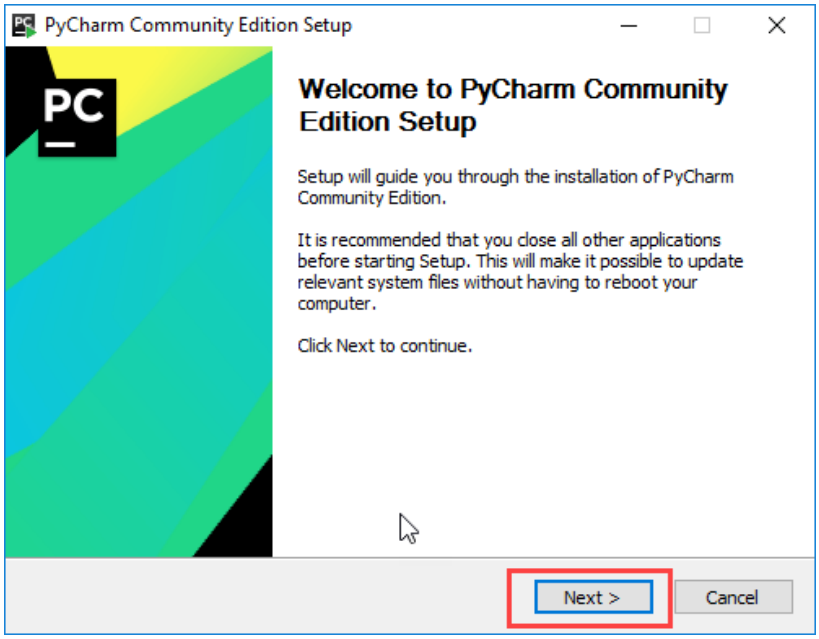
Go to https://www.jetbrains.com/pycharm/download/#section=windows and download the 'Community' version (circled in red).



Step 4 – Installing PyCharm (Our IDE)

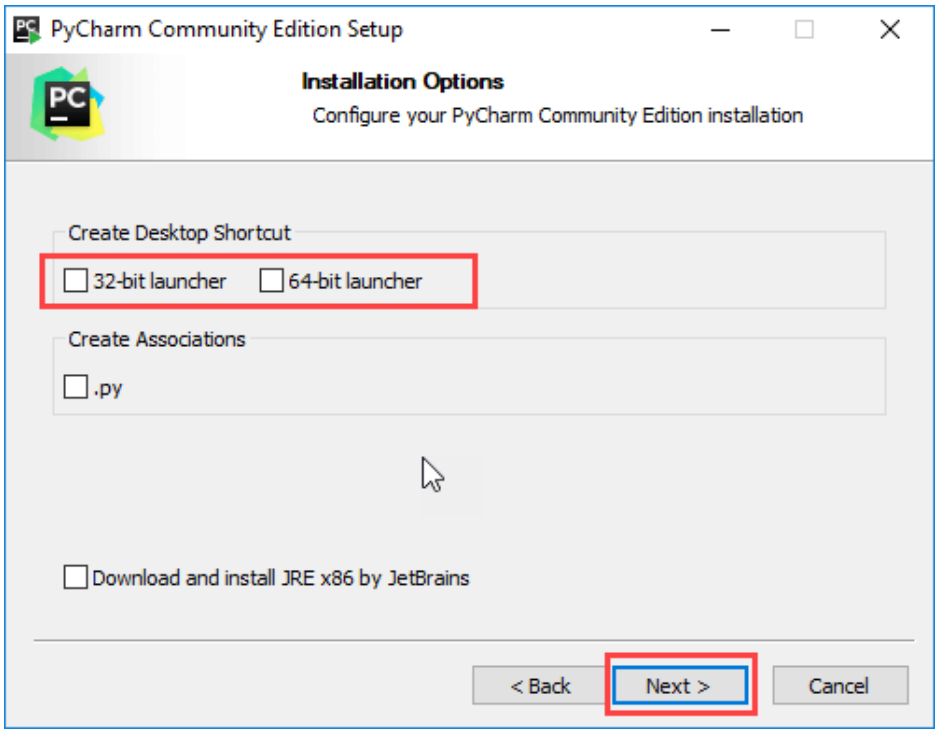
Once downloaded, find the file and run it. An installation wizard should open (see below).

Select Next (circled in red).



If you want a desktop shortcut, choose '64-bit launcher' (if your machine is 64-bit otherwise select 32-bit) – Circled in red.

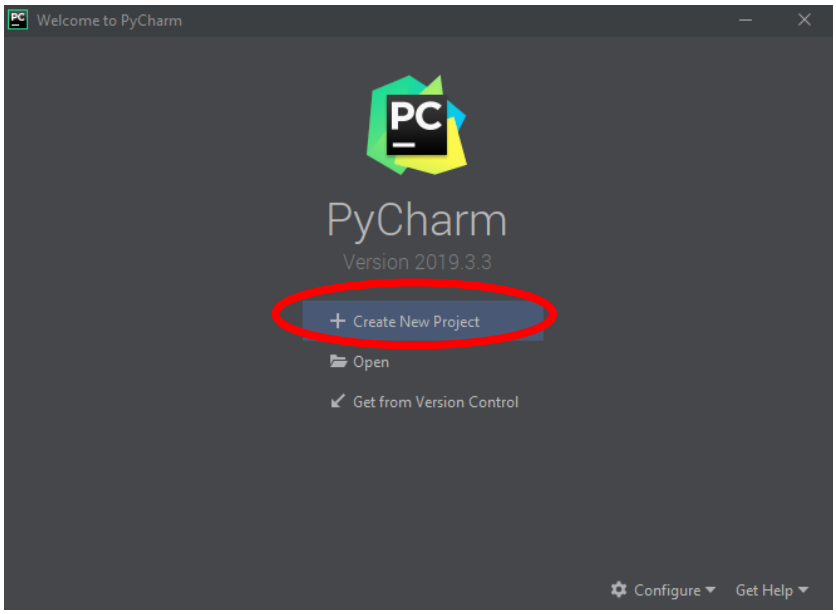
Tick the box to enable file associations with .py files too (circled in red).



Step 5 – Creating a Project

After installation, run PyCharm and you should be greeted to this window.

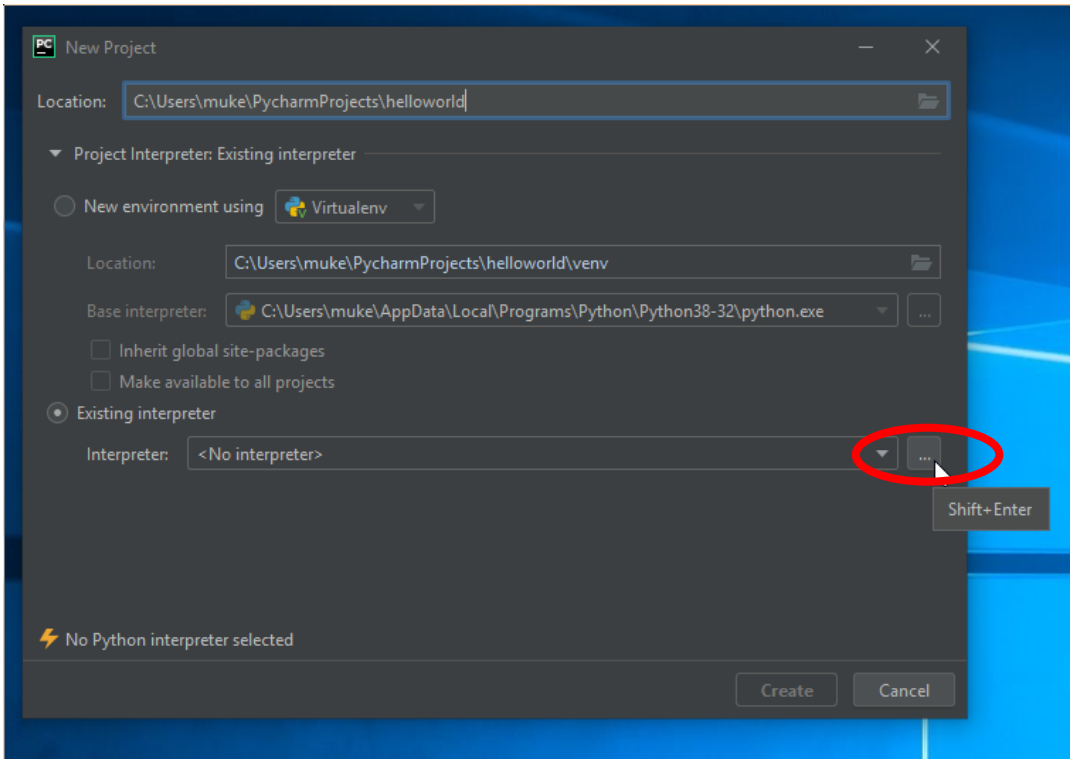
Click ‘Create New Project’ to get our first Python program off the ground!



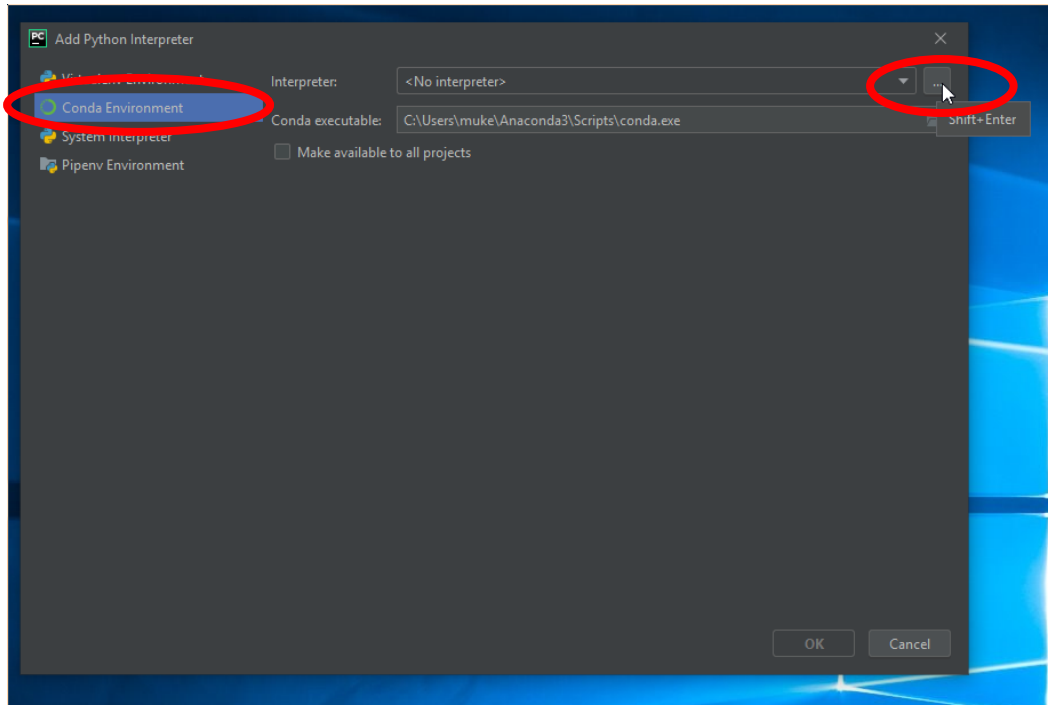
Firstly, PyCharm will ask you to select which interpreter to use.

As mentioned at the start, in our case this will be the Python interpreter that comes with the Anaconda package.

Select ‘Existing interpreter’ and press the ‘...’ button (circled in red).



You should then be presented with the window/dialogue box below. Navigate to the ‘Conda Environment’ tab, then press the ‘...’ button to navigate to where the Anaconda package has been installed so we can specify our interpreter. All circled in red.

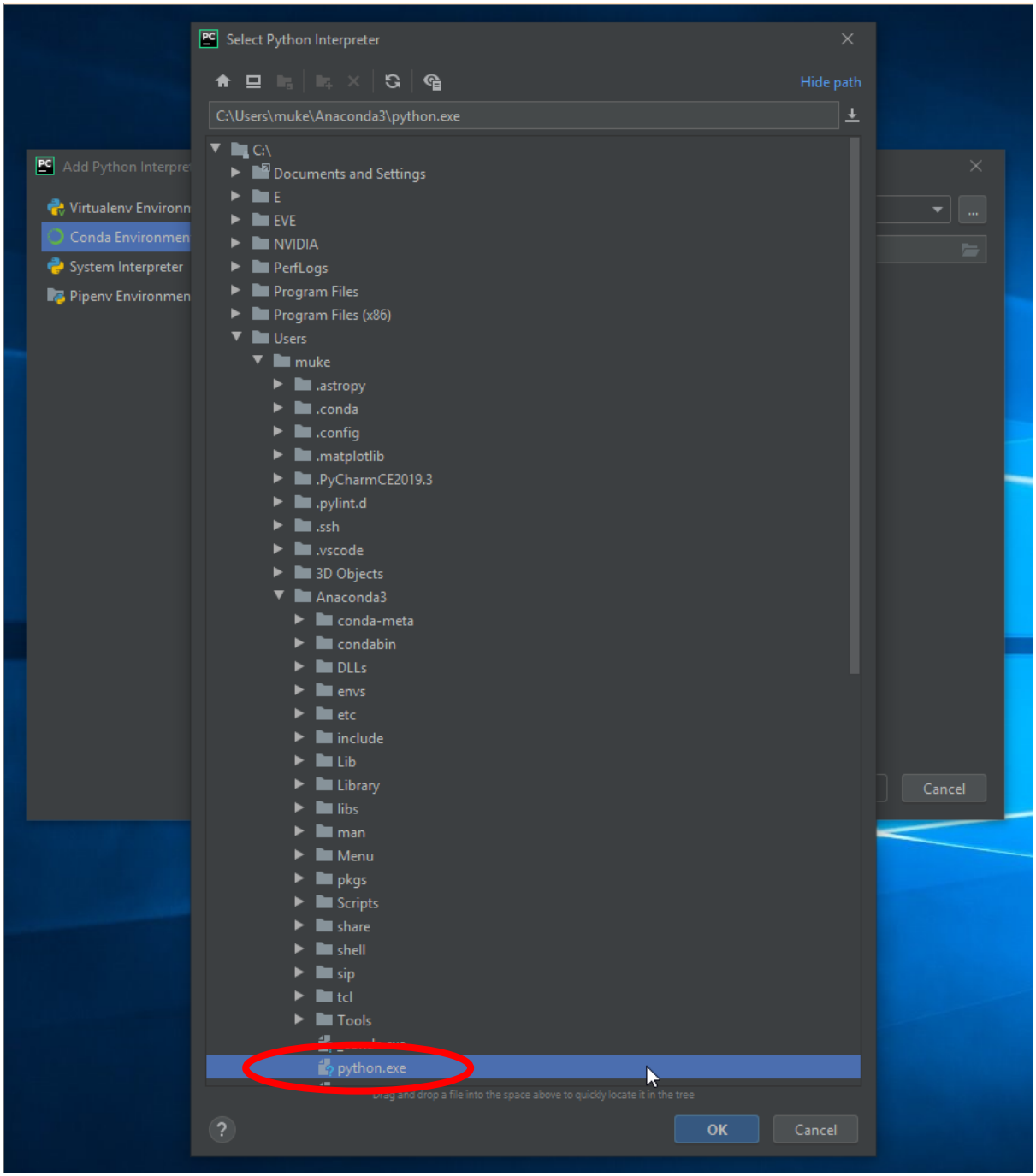


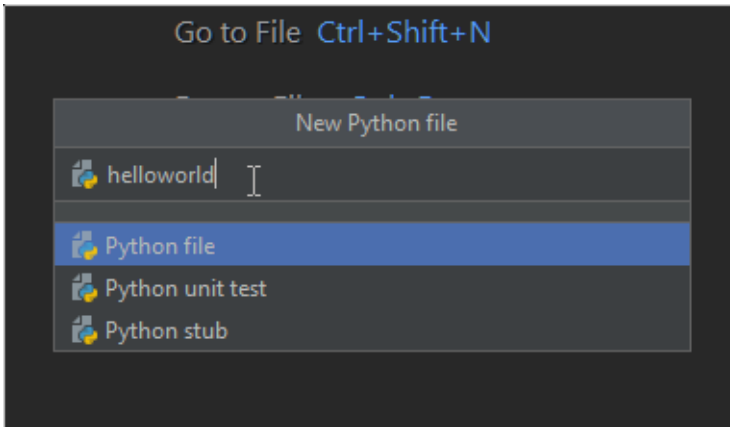
You will then be presented with the window below.

Your Anaconda will likely have been installed under the following directory: C:\Users\<username>\Anaconda3.

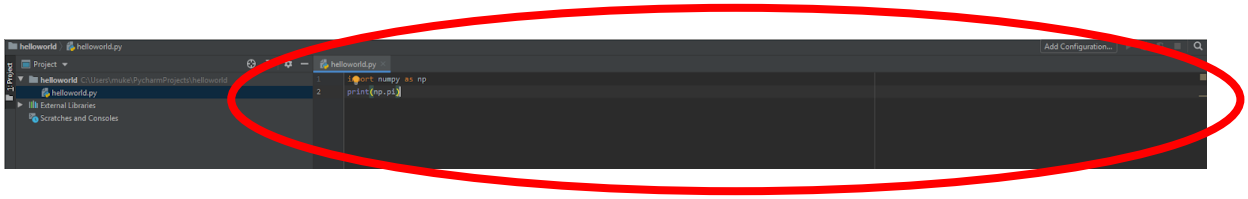
In this folder, we want to select ‘python.exe’ as our interpreter (circled in red).

Once this is selected, we can press ‘create’ for our project on the original PyCharm window.





You should now be presented with the screen below. And now we can start coding! The area circled in red is where you can type your code.



Write whatever you want in this field - if you want to try out a package like numpy, you can follow what's done above.

If you just want plain Python (if you're unsure, this'll be you!), you can simply write 'print('hello world!')' on the first line.

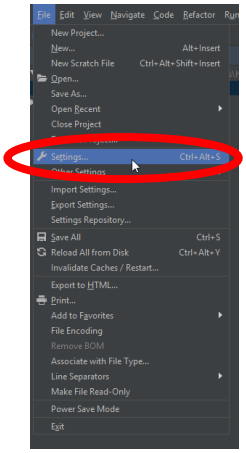
The next steps will go over how to install and enable external packages in PyCharm such as:

1. NumPy
2. SciPy
3. Pandas

Again, if you are not intending to use these, you can skip these steps and come back to them later when you do need those packages.

Step 6 – Importing Packages (NumPy) [OPTIONAL]

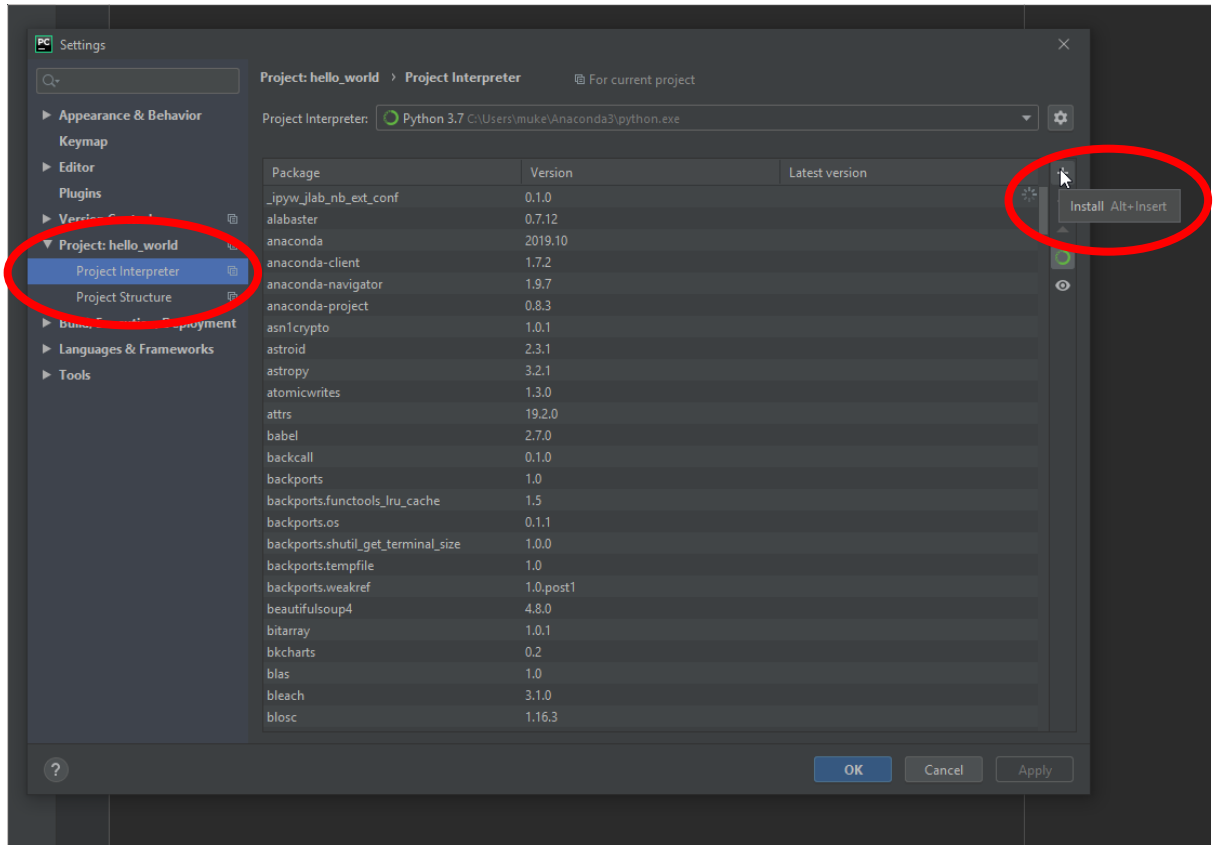
In the PyCharm IDE click “File” in the top left hand corner and navigate to settings (circled in red).



You will then be presented with the window below.

Go down to ‘Project: <project name>’ and under that ‘Project Interpreter’. We named our project “hello_world" which is why the dialogue box states “Project: hello_world”.

On the new window that comes up, go over to the right and click the plus button to select a package to install. All circled in red.



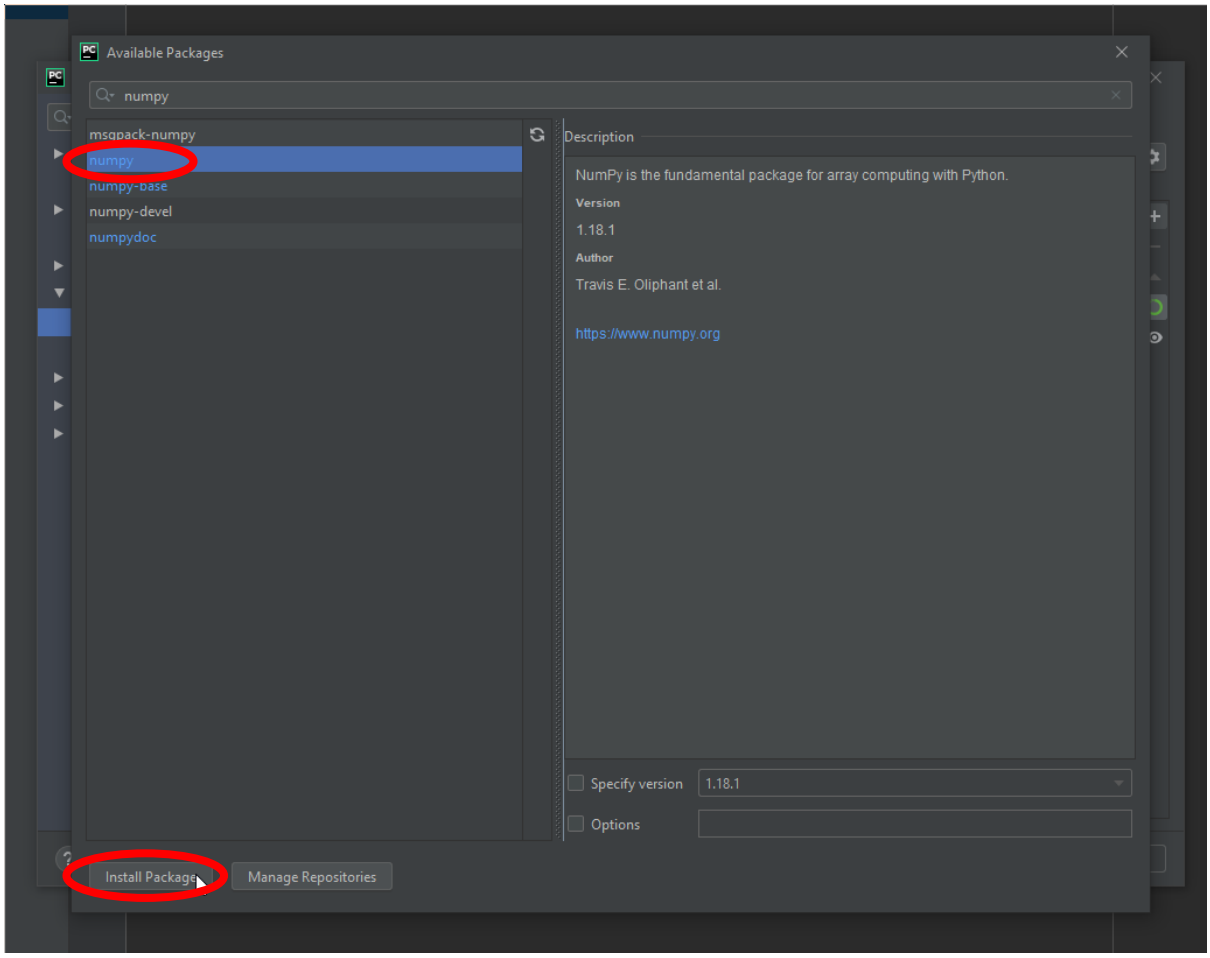
A new window will come up in which we can search for the package we want – as seen below.
For the purposes of this tutorial, let's use NumPy.

Search for 'numpy' and select the option we want.

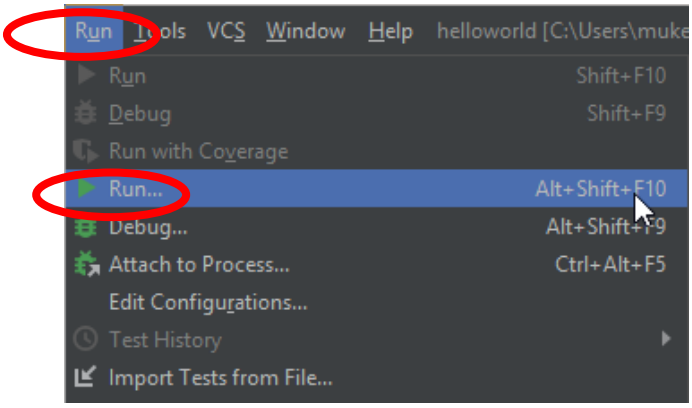
Then go to the bottom of the window and click 'Install Package'.

PyCharm will now install the package for you automatically from the internet. You will have to wait a few moments for this. There should be a progress bar of some sort at the bottom right of the main PyCharm window.

All circled in red.

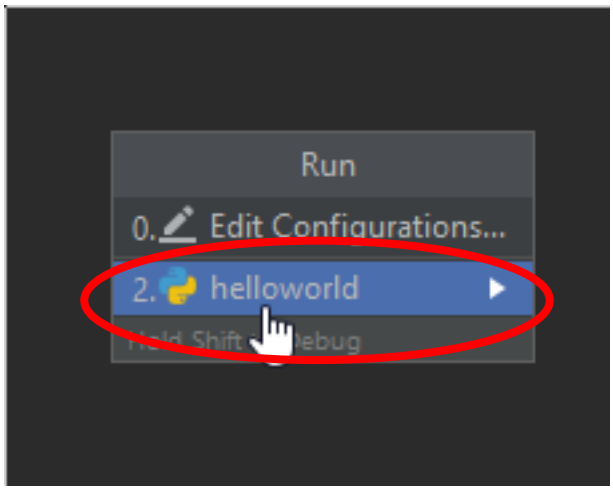


Now we are ready to “run”.
For our first run, we need to go to the toolbar at the top of PyCharm and select ‘Run’.



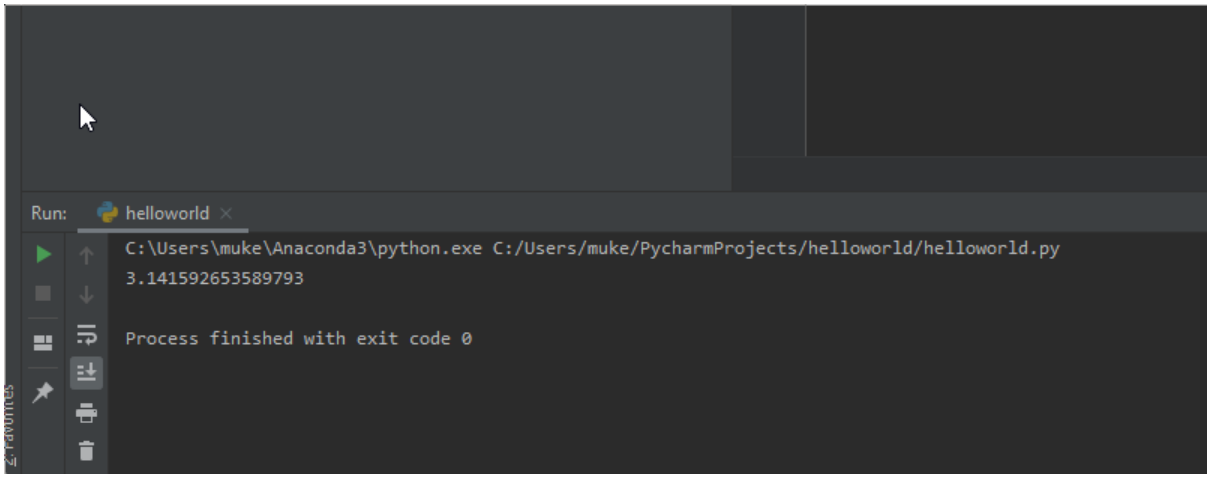
You will then be presented with a list of available Python files that can be “ran”. We only have a single Python file in our project called “helloworld”.

Select “helloworld” (circled in red).



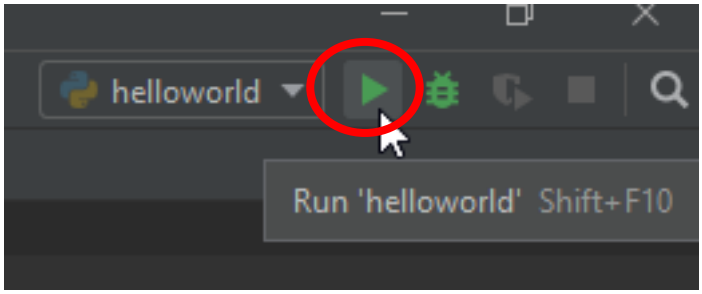
And there you have it! A new terminal will pop up at the bottom and output the result of running our program. In this case, as we wanted to test out using NumPy, we imported the numpy package and printed out a variable that comes with that package, which we can see to be the first 10 digits of Pi. If you opted not to install any package for now, you should just see 'Hello World!', or whatever you decided to print out!

We now have a fully functioning Python environment that we can program whatever want in.



For future reference, after the first run of our program, we get the option to run Python without having to go to the toolbar.

Instead, a new run button will no longer be greyed out at the top right hand side of PyCharm. It'll be easier to press this directly from now on!



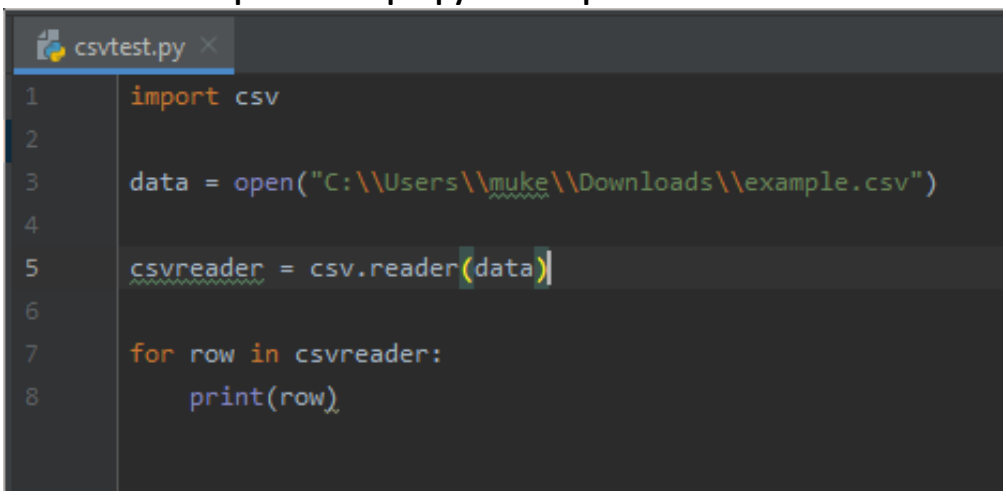
Reading Data from a .CSV File

Introduction

This guide will walk you through extracting data from a .CSV file[3]. There will be many scenarios where you will be required to process datasets stored in a file. For this you'll need to extract the contents of the file into our code so we can process and transform the data.

Find the example .CSV file used here: https://github.com/muke101/learnr-solutions/tree/master/Reference Book and Misc

Below is an example of a simple python script to read the contents of a .CSV file:

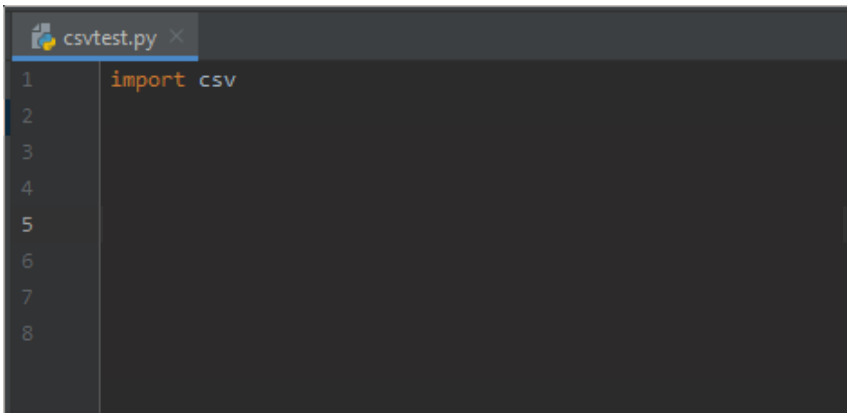


Let's go through this line by line.

[3] The CSV (Comma Separated Values) file format is the most common import and export format for speadsheets and databases

Step 1 – Import the “CSV” Package

First type in “import csv” (see line 1 below).



Step 2 – “Opening” the .CSV File

Now that we have imported the CSV package, we will be able to use a number of operations to allow us to process the file.

On line 3 declare a variable to assign the source of the file to. In our case we have named our variable “data”.

Next we need to “open” the file, so we use the function “open()”. In the parentheses specify the file path where the .CSV you wish to read is located.

In our case, the file is located at:
`“C:\\Users\\muke\\Downloads\\example.csv”`

PLEASE NOTE: Back slashes are reserved/special characters in programming, therefore, to use them as simple text values (like in the file path below) we have to “escape” them. This is very easy to do, simply add an extra backslash. The extra backslash tells the computer that we don’t intend to use the special functionality of that character and instead just want its text representation.

BEST PRACTICE: Add an argument ‘mode=r’ at the end of the open function call to specify you are reading it and not writing to it, but this is not necessary in python.

See below for code:
open(<file path>, mode=’r’)

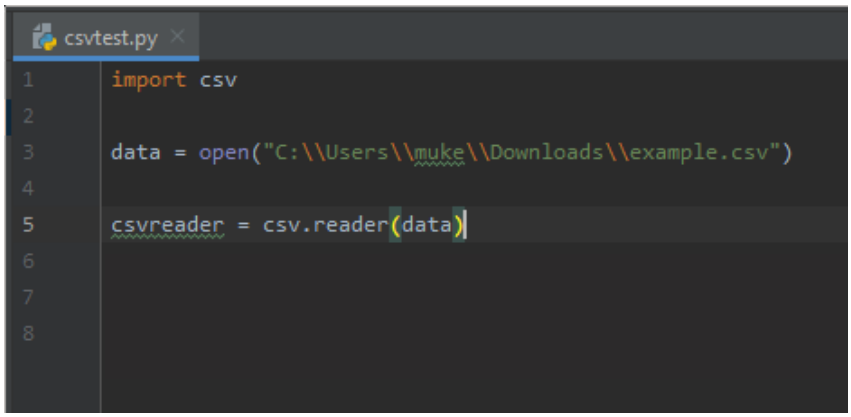
Step 3 – Creating the Iterator

Now on line 5 create a variable that will form our “reader” of the data that has now been extracted to allow for meaningful processing.

The type of this variable is called an ‘iterator’. We use a function from the csv package called “reader()”, the argument supplied to the function is the source of the data (variable = data).

The iterator will return us new values from the data source we specified as we ask for them on the fly.

Don’t worry too much about the specifics of this though! It just means we can only process data as we iterate over the data source. We can’t pinpoint data and “call” it like we can with a variable.



Step 4 – Extracting & Printing the Data (Each Row As List)

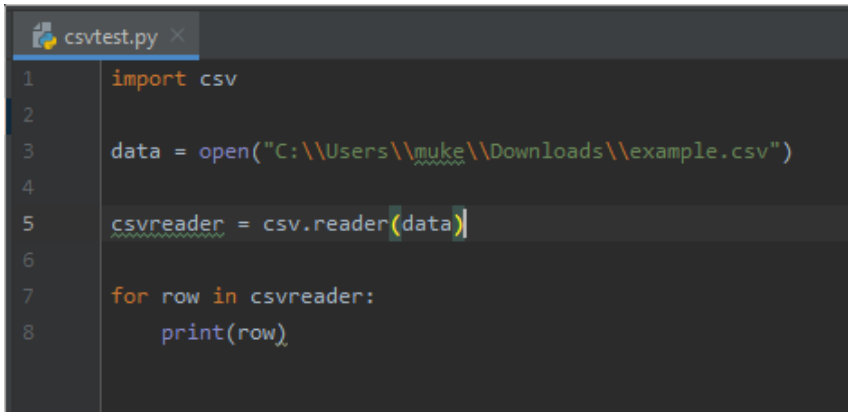
The data in the .CSV file is stored line by line; we therefore extract the data line by line.

For arguments sake lets visualise that the .CSV file contains a table or data (headings, values). The first line will be the headers, the lines proceeding that will be a record or row in that table. This is what the iterator variable is used for, to extract line by line.

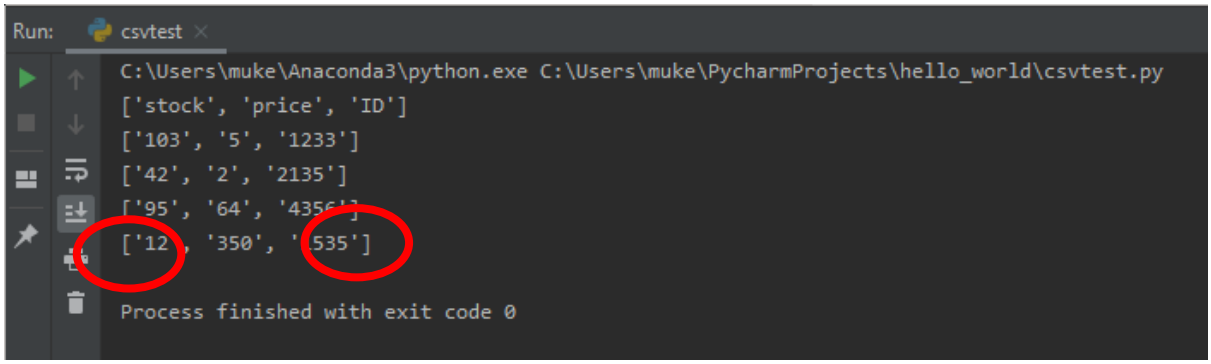
Now let's begin the extraction itself.

As shown on line 7 below create a for-loop, select a loop variable for use within the loop code block – in our example we have chosen "row" to represent the row of data extracted from the .CSV file. Every time the loop executes a new row is extracted from the "csvreader" iterator variable and passed to the loop variable "row" – the loop stops when there are no more lines.

On line 8 we pass the loop variable as an argument to the print() function to print the row to the console.



As you can see below we have the contents of our file! If you look closely notice the square brackets (circled in red) during processing each row containing our values has been loads into a Python list. This will allow further manipulation of the data.



Step 5 – Extracting & Printing the Data (Each Value in File on Separate Line)

Perfect! It's a simple step from here to print out each individual value in the file on its own line too. If you remember to the previous step each row extracted is "typed as a list" or in other words the data type is a list. A list can also be "looped" over so as shown below simply create another for-loop within your previous for-loop (see below).

This is referred to as nested for-loops, which means a for-loop within a for-loop. This nesting allows us to first extract a single line/row/record from our data, this is of list format and is passed to loop variable "row".

As the loop variable "row" is of type list, we are able to use that to create another for loop but focused to just a single row, which contain the values and extract them one by one. Create your nested for-loop as shown on line 8, select a loop variable in our case it's "entry" – the loop stops when there are no more lines.

On line 9 we pass the loop variable as an argument to the print() function to print the value to the console.

REMEMBER
OUTER FOR-LOOP: Extracts a single line
NESTED FOR-LOOP: Extracts a single value from the extracted single line

```
7    for row in csvreader:
8        for entry in row:
9            print(entry)
```

```
csvtest
C:\Users\muke\Anaconda3\python.exe C:\Users\muke\PycharmProjects\hello_world\csvtest.py
stock
price
ID
103
5
1233
42
2
2135
95
64
4356
12
350
1535

Process finished with exit code 0
```

Writing Data to a .CSV File

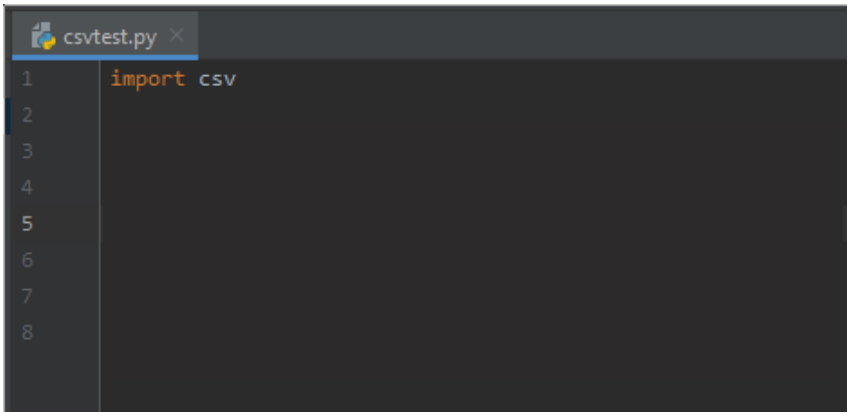
Writing to a .CSV file is as easy as reading from them. Find the example .CSV file used here: https://github.com/muke101/learnr-solutions/tree/master/Reference Book and Misc

Below is a script which writes to our .CSV file. Let's go through it line by line again:

```
1  import csv
2
3  data = open("C:\\Users\\muke\\Downloads\\example.csv", 'a')
4
5  csvwriter = csv.writer(data, quoting=csv.QUOTE_MINIMAL)
6
7  csvwriter.writerow(["123", "456", "789"])
8
```

Step 1 – Import the "CSV" Package

First type in "import csv" (see line 1 below).



Step 2 – "Opening" the .CSV File

Now that we have imported the CSV package, we will be able to use a number of operations to allow us to process the file.

On line 3 declare a variable to assign the source of the file to. In our case we have named our variable "data".

Next we need to "open" the file, so we use the function "open()". In the parentheses specify the file path where the .CSV you wish to read is located.

However, unlike last time we have to add the argument 'a'. This argument specifies we want to append to the end of the file we have "opened".

A NOTE ON MODES: We could also use the 'w' argument to specify we want to either *write* to a new file (and thus create one at the filepath specified) or overwrite an existing file from scratch at the specified path. This will delete the existing data of a file we give it without warning though, so be careful and use the 'a' flag if you just want to add data instead of write from scratch!

In our case, the file is located at:
`"C:\\Users\\muke\\Downloads\\example.csv"`

PLEASE NOTE: Back slashes are reserved/special characters in programming, therefore, to use them as simple text values (like in the file path below) we have to "escape" them. This is very easy to do, simply add an extra backslash. The extra backslash tells the computer that we don't intend to use the special functionality of that character and instead just want its text representation.

```
1
2
3
4
5
6
7
8
```

```
import csv

data = open("C:\\Users\\muke\\Downloads\\example.csv", 'a')
```

Step 3 – Creating the Writer

To write to a .CSV file we cannot use the 'reader()' function from the csv package. Instead we have to use the 'writer()' function.

Sometimes the additional 'quoting' argument is helpful, but usually it's not necessary. On the most basic level this specifies treatment of fields containing special characters such as delimeter, quotechar, or any of the characters in lineterminator.

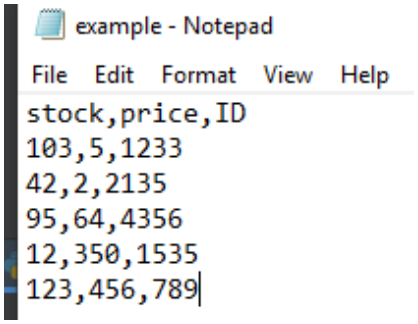
```
1  import csv
2
3  data = open("C:\\Users\\muke\\Downloads\\example.csv", 'a')
4
5  csvwriter = csv.writer(data, quoting=csv.QUOTE_MINIMAL)
6
7
8
```

Step 4 – Writing to the File

On line 7, is where we write our data to the .CSV file itself.

We call the 'writerow()' function. This function call will create a new row in the given file. The function takes a *single* argument in the form of a list of *multiple* values.

We can't pass "123", "456", "789" as separate arguments, we must give them to the 'writerow' function as a list and then Python will split them up in the .CSV file for us automatically to create the new row.

```
1  import csv
2
3  data = open("C:\\Users\\muke\\Downloads\\example.csv", 'a')
4
5  csvwriter = csv.writer(data, quoting=csv.QUOTE_MINIMAL)
6
7  csvwriter.writerow(["123", "456", "789"])
8
```

Let's take a look at our original .CSV file now that we've written to it, as you can see the last row is the one we just appended (see below).