Common python IDEs

1. PyCharm

2. Jupyter Notebooks

3. Atom text editor

4. Spyder

5. PyDev (Eclipse)

6. Sublime Text 3

7. Visual Studio

Most commonly used python libraries: Numpy, Pandas, Scipy, Scikit-learn and Seaborn, mplotlib

Name of library	**Primary Purpose**
Pandas	Data cleaning and manipulation, similar to dplyr in R
Pygames	To create games (similar to scratch)
Seaborn	For data visualization
matplotlib	For Data Visualization
Plotly	For Data Visualization
Scikitlearn	For machine learning
Django	Creation of webpages

Some General Advice

- Always code using the IDLE interface as it lets you change your code if you make a mistake
- On the idle interface, go to file and click “new”.
- Type your code here
- Save it wherever you want with a .py extension. E.g. mydata.py
- Then go to run and click run module. The output for all the commands will be displayed on the IDLE. If you make a change, just save and run the entire thing again.
- This is similar to the function of a do-file in Stata

(Python Shell)

Command	**Function**
print(“hello world”)	
for x in range(10):	

print(x,x*0.5)	

The Jupyter interface

What are Jupyter Notebooks: they are a way of writing code. To access it, download python, go to search and type “Jupyter”. A new tab will open. Create a new python3 notebook and start coding

- If you download Anaconda from their website, you will automatically get both python and Jupyter
- Type your command in the input field and press shift and enter simultaneously to run in it
- Pressing A adds a new input field above your existing input field
- Pressing B adds a new input filed below your existing input field
- Selecting an input field and pressing D twice deletes it
- There are four types of data on Jupyter; string, integer, float (has decimals) and Boolean (True and False)
- Much like RStudio, any text following # will be treated as a comment and not as code
- Like Stata, if you want to indent a line (if the command is too long). Add / and continue typing your code in the next line
- When indexing elements, count from zero and not one. HOWEVER THIS DOES NOT APPLY WHEN YOU ARE COUNTING BACKWARDS (last to first)
- Only text typed immediately next to the left-hand side text border will produce an output, if any text is indented, no output will be produced
- + can be used to add strings together. E.g. ‘string1’+’string2’
- Use the tab key on your keyboard to create indentation
- The or operator needs AT LEAST ONE true statement to return true (as output)
- The and operator needs ALL statements to be true to return true (as output)
- If multiple not/or/and are used with Booleans in the same line of code; the order to read and interpret them is as follows: not, and, or
- When using if, else commands, you can only use two conditions; one for if and one for else. Multiple elif conditions can be used however and you can create as many conditions as you like. The order of conditions are if, elif (how many ever you want) and finally an else condition
- A colon needs to be included after each if, else, elif statement
- A colon has to be added after functions and loops as well
- Suppose you have two elif commands such that
  Elif x>0:
  Return “a”
  Elif x>2

Return “b”

- You then print the value x==4. What happens? Clearly both conditions are satisfied. The value a will be returned as that is the one that gets satisfied FIRST. So Jupyter doesn’t bother checking the subsequent commands as the conditions are already satisfied
- Return can only be used once in a function
- When print is being used in a function; it does not affect the calculation of the output. It only prints out statements
- When return is being used in a function; it does not visualize the output but rather specifies what that function is supposed to give back as output. E.g. a+10 or a-5 etc etc. Return cant be used to print statements but rather give out a value of an a variable
- If you are planning to use both print and return in a function, use the print command first. Else it wont work
- When making a list of strings, place the elements in a square bracket and wrap quotation marks around each one of them
- Lists can store strings, floats and integers
- When slicing (sub setting) a list; list the index number of the starting element you want to slice and ONE PLUS THE INDEX NUMBER OF THE LAST element you want to slice. E.g. if you want to slice elements 2 to 5 in a list called abc, you should type abc[2:6]
- Indexing is always done using square brackets
- The method .sort sorts strings in a list in alphabetical order and numbers in ascending order
- Tuples are like lists but they are immutable; they cant be sliced, appended on, modified. They are encased in round brackets (parenthesis) and not square brackets
- Tuples need not necessarily need a bracket. If you simply type x= 23,45,56; Jupyter assumes it to be a tuple. A tuple can also contain only element
- E.g. a,b,c=1,2,3. We have created three tuples; a, b and c which equal 1,2 and 3
- Tuples can be added as elements within a list. Each Tuple becomes an element in that list

- Functions can provide tuples as return values. Which is useful because functions can normally return only one value otherwise; but thanks to Tuples; they can return more than 1
- Dictionaries are a way of storing values; just like Tuples and Lists. However, the difference is that in a dictionary; you get to create your own index numbers rather than getting it assigned.
- The self-created index numbers are called ‘keys’. So to call an element, you can use the key instead of the index number
- A key can correspond to one or more elements
- Keys should be referred to using single inverted commas
- Lists have square brackets, tuples have round brackets (Paranthesis) and dictionaries have squiggly brackets
- Incrementing: Adding a number on top of an existing variable during is a loop is called incrementing e.g. x=x+1
- In the range function, the ‘stop’ number is the last value in the list+1
- In the range function, the step number is the difference between the two consecutive numbers
- In the range function; the start and step numbers are assumed to be 0 and 1 if not entered. Stop number is compulsory

- In mathematics, composite functions are read from right to left e.g. g(f(x)) means that the function f is first applied to x and then g is applied to f(x). The exact same logic applies in Python as well. Suppose you have the following

x=[0,1,2]

what would range(len(x)) be?

First we start with len. len(x)=3 as there are three elements

range(3)=range(0,3,1)=start from 0, end at two, change by +1 each time. so range(len(x))=[0,1,2]

- Each object/instance in Python belongs to a certain class ; defining the rules for creating that object. We can even attach attributes/properties to that object
- Two more differences between a function and a method; a function can have more than one parameter whereas for the method, the object that the method is being applied on is the parameter. That brings us to the second difference; a function can exist on its own whereas a method needs an object to exist. That is why the syntax for a function is function() but for a method, it is object.method. This object can be a list/tuple
- In python; a module is pre written code containing definitions of functions and classes whereas a module is a collection or directory of related modules
-

Using a Jupyter Notebook (python 3.7)

Command	Explanation
X=5	You have defined a variable called x which takes the value 5
X=5**2	You have defined a variable called X which is the square of 5.
X=pow(5,2)	You have defined a variable called X which is the square of 5
X	If you now type X, the output will display 25
X=(2,3,4,5,6)	We have defined a sequence of numbers called x
Y=len(x)	We have a defined a new variable called Y which should you how many elements there are in sequence X
Y=len(“Sravan”)	We have defined a new variable called Y which tells us how many letters there are in “Sravan”
max(x)	Returns the largest number in the sequence. This result can be stored in a variable
min(x)	Returns the smallest number in the sequence. This result can be stored in a variable
abs(x)	Suppose x is a single number. abs(x) gives us the absolute value of x
X==25	Jupyter reads this as is X==25? In this case, it is because 5 squared is 25. Therefore, Jupyter will return True (Boolean)
X==26	Jupyter reads this as X==26? In this case it is false because X=5^2 which is 25

X!=26	Jupyter reads this as X is not equal to 26? In this case, it is true because X is 25. So, the output returned will True
15>10+10	Jupyter reads this is 15 greater than 10+10? As it isn't, the answer returned will be False
15<=10+5	Jupyter reads this as is 15 less than or equal to 10+5. As it is equal to 10+5, the answer will be True
3>5 and 10>5	You have typed out two statements. The second is true but the first one is false. The and operator requires BOTH statements to be true. So, the output printed will be false
3<5 and 10>5	Here both statements are true so the output returned will be true
Int(x)	Converts x into an integer
Str(x)	Converts x into an string
Float(x)	Converts x into a float (decimal)
type(x)	Tells you what x is; boolean, string, integer or float
"Friday"[3]	Extracts the fourth letter of the string Friday and prints it out. In this case it is d. As we are counting from zero, d is chosen instead of i . use square brackets around the index number

if 5==15/3:
    print("hooray")
	Prints the word hooray if 5=15/3. The print command should be indented. If 15/3 does not equal 3, no output is returned. Multiple print commands can be used

x=2
if x==2:
    print("right")
else:
 print("wrong")
	This time we have specified an else statement. So an output will have to be returned. Once again, not that the print commands have to be indented.
	
	If and else have to be placed on the same vertical line.

def function1(y):
    if y>5:
        return "g"
    elif y==5:
        return "e"
    else:
        return "l"
	We have defined a function called function function1 which takes the value y. If y>5 we want jupyter to return the value g, if it is equal to y, it will return e and l if less than y.
	
	The if, elif and else commands are on the same vertical line
	The if, elif and else commands are one indent away from the def function whereas the return command is two indents away from the def function

print(function1(5))	This command prints the function, y=5. As per our rule above, we should get the value e.

def simple():
 print("hey")
	You have defined a parameter less function called simple that prints hey when typed out. To get the output "hey", type out simple(). This function is parameter-less because there is no parameter in the bracket after the function (simple)
	
	When typing the print command, use the tab key to indent

def plus_ten(a):
      return a+10
	A function called plus10 has been defined wherein a is a parameter.
	We want Jupyter to return the value which is ten greater than that number

Plus_ten(2)	We should get a value that equals 12

def plus_ten(a):
      x=a+10
     return x
	This is a slightly longer way to create the same function listed above. We have created a variable called x which stores the value a+10. So instead of returning a+10, we are returning x (which is basically the same thing)

def plus_ten(a):
    x=a+10
    print(“outcome”)
   return x
	In addition to returning the value of x, this function will now print the word “outcome” every time it is run

def abc(x):
   return x+5
def abd (abc)
   return abc*2
	We have defined two functions here; abc and abd. Abd is a function of the parameter abc (which is already a function). Therefore, abd is a composite function

abc(2), abd(5)	When you type this command; the values 7 and 14 will be returned. 7 is the value abc will taken (5+2) whereas 14 is the value that abd will take (7*2)

def new_function(x):
  if x>=100:
    print("case 1")
    return x+10
  else:
    print("case 2")
    return x-10
	We have a defined a function called new_function with a parameter x. If x>100; we will get the output x+10 and the words case 1 will be printed out. If x<100; the output x-10 will be returned and case 2 will be printed

def multiple(a,b,c):
  return (a+7)*b*2*c
	We have defined a function called multiple with three parameters; a,b,c.

multiple(2,3,4)	When we type this we get the output 216. (2+7)*3*2*4. In this case; a=2, b=3 and c=4
Round(3.567,1)	Rounds 3.567 into 1 decimal place. This result can be stored in a variable
participants=['John', 'Joshua', 'Katie']	We have created a list consisting of three separate strings; John, Joshua and Katie
(participants[1])	Prints the string Joshua. Remember that python starts counting from zero. So 0=John, 1=Joshua and 2=Katie
Participants[-1]	Prints the last string in this list. That is Katie
Participants[-2]	Prints the second last string in this list. This is Joshua
Participants[2]=’Maria’ Participants	We want to replace the string Katie with Maria. Remember that katies index number is 2. Simply type Participants again to get the new list
Del Participants[0] Participants	We want to remove the String John from the list. So we delete it using the command on the left. Note that the list only contains two names now; Joshua and Maria and their index number changes because of the deletion. So now, Joshua is 0 and Maria is 1
Participants.append(“Dwayne”) Participants	The .append lets us add strings to a list. The string Dwayne has been added to our list. Dwayne will appear at the end (index number 2). Notice that Dwayne is in double brackets here
Participants.extend([‘Christina’, ‘George’]) Participants	The .extend also lets us do the same thing; though the syntax is different here. We put the new strings in square brackets which are then inside normal brackets. Christina and George will be added to the end of the list. So Christina will have an index number of 3 and George will have 4
print('The second participant is ' + participants[1]+ '.')	The statement” The second participant is ‘Dwayne’ will be printed out’. Notice that we don’t need to put participants[1] in quotes.

Participants= [‘A’,’ B’,’ C’,’D’,’E’}
xyz=Participants[1:3]
pint(xyz)
	We have defined a list consisting of the elements, A,B,C,D,E.
	Xyz is a vector that consists of the the letters B and C (second and third elements of the list)

Participants= [‘A’,’ B’,’ C’,’D’,’E’}
xyz1=Participants[:3]
pint(xyz1)
	We have extracted all elements up until the third element. So if you print xyz1; you will get A, B and C

Participants= [‘A’,’ B’,’ C’,’D’,’E’}
xyz1=Participants[:-2]
pint(xyz1)
	This will give us the exact same result as above

Participants= [‘A’,’ B’,’ C’,’D’,’E’}
xyz2=Participants[2:]
pint(xyz2)
	We have extracted all the elements starting from the (1+1) the 2nd element. So if you print out xyz2; you will get C, D and E

Participants= [‘A’,’ B’,’ C’,’D’,’E’}
xyz2=Participants[-3:]
pint(xyz2)
	This will give us the exact same result as above

Participants.index(“B”)	Suppose you know that B is in your list but you don’t know the indexnumber, this command will tell you! Use normal brackets and not square brackets

List1=[‘a’, ‘b’]
List2= [‘c’, ‘d’]
List3=[List1, List2]
	We have created two lists called list one and list two and combined them together to create a bigger list called List3.

Participants= [‘A’,’ B’,’ C’,’D’,’E’}
Participants.sort()
Participants
	Sorts the list called Participants in alphabetical order. Had this list consisted of numbers, they would be sorted in ascending order. Print the list again to see it in order

Participants.sort(reverse=True) Participants	Sorts the list in reverse alphabetical order or descending order

x=3
y=[1,2,x]
y
	We have defined a tuple called x and added it to a list called y.
	So when y is typed, the output will be 3

X=(1,3,4,5) X[0]	We have defined a tuple called x and called the first element. The output returned will be 1
Srav={‘k1: “dog”, ‘k2’: “cat”, ‘k3’: “rat”} srav	We have created a dictionary called Srav. This dictionary contains the elements, dog. These elements have the keys (index numbers); k1, k2, k3
srav[‘k1’]	This returns the element associated with k2; in this case dog
srav[‘k4’]=crow	We have added a new element crow into the dictionary named srav. The key associated with this element is k4
srav[‘k2’]=”kitten: srav	We have replaced the element associated with the key k2 from cat to kitten. So when we type srav again, this change will be reflected
srav={‘k1: “dog”, ‘k2’: “cat”, ‘k3’: “rat” ‘k5’: [“rabbit”, “fox”]} Srav[k5]	It is possible to have one key for more than one element. Here k5 is associated with two elements; fox and rabbit. So when we run the second line of code; the output fox and rabbit is returned K5 can be thought of as a list
Team={} Team[‘striker’]=”John” Team[‘Defender’]=”Sam” Team[‘Centre’]=”Jake”	This is another way of creating a Dictionary. The dictionary is called team and the keys are called striker, defender and Centre. John, Same and Jake are the elements
Print team.get[‘ Centre’]	Prints the element associated with Centre

Team['Goalkeeper]="Ron"	We have added a new element to the dictionary. The key is goalkeeper and the element is Ron
Team['Centre']="Jack"	We have replaced Jake with Jack

B= [0,1,6,7,8]
for c in B:
      print(2*c)
	This is a loop which lets you to print two times each value in the in the list. The print command has to be indented

B= [0,1,6,7,8]
for c in B:
      print(2*c, end = " ")
	This command does the exact same thing but the elements are printed on the same line. The space in between the two double inverted commas implies that each element should have a space before the next element

x=1
while x<10:
    print(x, end=" ")
    x=x+1
	This is a while loop. First we specify a starting value for the variable x. Till the time x is smaller than 10; its values will be printed out. The last command: x=x+1 is needed to convert an infinite loop into a finite loop. If we were not incrementing the value of x; x would always be less tan 10 (x=1) and the number 1 would be printed endliessly

x=1
while x<10:
    print(x, end=" ")
    x+=1
	Does the exact same thing as the above line of code. Notice how we changed x=x+1 to x+=1. This is nothing but a change in syntax

range(10)	We have specified a range of numbers which will go from 0 to 9, increasing by 1 at each time. As we have not specified a start and step number, Jupyter assumes them to be 0 and 1 respectively
list(range(10))	When the list and range commands are combined; the numbers are printed out as a sequence. Note that the sequence stops at 9 as according to pythonic logic, the stop number is one more than the last number
list(range(5,20,2))	This will list as a sequence, numbers starting from five, (with a difference of two between two consecutive values) and ending at 19

for number in range(10):
    if number % 2 ==0:
        print(number, end= " ")
    else:
        print("Sorry", end= " ")
	We have combined an if and else statement with a loop. We have defined a sequence of functions 0-9 using the range function. Then we used an if condition to specify that if a particular number in that list divides cleanly by 2, without leaving a remainder (in other words, that number is even), then we want Jupyter to print that number
	
	If that number is odd (as described the else function); then we want the word sorry to be printed out. So we will have the output; 0 sorry 2 sorry 4 sorry 6 sorry 8 sorry

for number in range(3): print(number, end = " ")	Prints the numbers 0,1,2

x=[0,1,2]
for number in range(len(x))
      print(x[number], end =" ")
	This is a more advanced way of repeating the above step.
	We have defined a list of elements called x. We then want to loop over the range of the length of the list x. and print it. Notice that we had to index the "number" to the list x to extract and print each element in the list x.
	
	So we are essentially printing out x[0]=0 x[1]=1 and x[2]=2. Where the index numbers are given by "number" as defined in out second line.

List1= [0,12,3,4,25,69]
 	
def less20(x):
	Suppose we want to find out the number of elements in list1 which are less than 20? How do we do this?

y=0
for z in x:
if z<20:
y +=1
return y
 	
less20(List1)
	We combine loops, functions and a conditional statement to achieve our objective. We first define a function called less20. Then we generate a variable called y. y gets incremented by 1 if z is less than 20 for each value in that list. Then we can ask Jupyter to return y so that we can see how many times it has been incremented. The for command is one indent to the right of the def command whereas if command is two indents to the right of the def function. y (in the second line), Return and For should align
	
	Finally we substitute list1 as the argument of the less20 function to see how many numbers in List1 are less than 20

prices={'egg':4, 'milk':6, 'bread':10}
quantities={'egg':12, 'milk':9, 'bread': 13 }
 	
money_spent=0
for b in prices:
money_spent = money_spent + (prices[b] * quantities[b])
 	
print(money_spent)
	We have two dictionaries called prices and quantities. Both have the same keys. The list the price and quantity purchased of three groceries. Now How do we find out the total expenditure
	
	We generate a new variable called money_spent. Then we start a loop where b takes the value of each key in prices (it can be quantity also; as the keys are the same). Now for each key in prices, it gets multiplied by the corresponding key in the quantities dictionary. Again, this is possible because both dictionaries have the same key
	The product of this number gets added to the money_spent variable and this goes on till each price-quanitity key gets multiplied and added to moneyt_spent
	
	Once we print money_spent, we get the total amount of money spent

help(name of command)	This command provides some guidance on how a particular command can be used

How to use Pandas

Code	Explanation

Import pandas as pd	We have imported a package called Pandas but we have renamed it as pd whilst importing it. So we only need to refer to it as pd here on in
Import DataFrama from pandas	Imports only the command DataFrame from Python. This means we need not type pd.DataFrame but simply dataframe
Import DataFrama from pandas as d	We have not only imported the DataFrame command but we have renamed it d. so we can simply type d(). Once again, we don’t need the the pd. prefix

1. http://www.danielmsullivan.com/pages/tutorial_stata_to_python.html: a stata to python dictionary
2. http://www.blog.pythonlibrary.org/2018/10/09/how-to-export-jupyter-notebooks-into-other-formats/: exporting jupyter notebooks to pdf
3. www.w3resource.com/p : A website containing sample exercises and their solutions in python

- Some Links specifically for Seaborn and Matplotlib (and data visualization in Python)

1. https://towardsdatascience.com/matplotlib-vs-seaborn-vs-plotly-f2b79f5bddb : A comparison of plotly, seaborn and matplotlib
2. https://blog.magrathealabs.com/choosing-one-of-many-python-visualization-tools-7eb36fa5855f : comparing various packages for data visualization

- Some Links Specifically for Pandas

1. http://hamelg.blogspot.com/2015/11/python-for-data-analysis-part-19_17.html: How to tabulate variables using Pandas
2. https://pandas.pydata.org/ : A website which has the Documentation of Pandas Commands
3. http://www.blog.pythonlibrary.org/2018/10/09/how-to-export-jupyter-notebooks-into-other-formats/: