*Check the accompanying Jupyter notebooks and .py scripts with many more examples and concepts*

**Programming**

* Source code must be written so the machine can process it
* Program: a sequence of instructions that designate how to execute a computation
* Programing: taking a task and writing it down in a programming language that the computer can understand and execute
* Many programming languages
  + Php: web programming
  + C++ : programming devices
  + Python and R: data science and finance
* A good programmer needs:
  + Problem solving skills
  + Abstract thinking
  + Mechanistic thinking
  + Solid knowledge of the syntax
  + Good coding style
* Why Python
  + Technical advantages
  + Wide practical application
  + It is an open-source, general-purpose, and high-level programming language
* Why Jupyter
  + It is a server-client application that helps use python to communicate with the machine
  + Kernels are the languages (Python, R, Julia) and interfaces are the clients
  + It is used because it offers access to different kernels in one space
  + Because it is a notebook, it facilitates communication
  + It has Text, code and output all on one page

**Python syntax, variables and data types**

* Variables are used to store information and present a data input
  + The line of code is called a command/program
  + = in Python means "assign"
* Data types
  + Type(var) to find the variable type
  + Type(value) to find the type of a value
  + Int(var) or int(value) turns it into an integer if it was another value (like float) before
  + Float(value) will add a decimal point and turn it into a float
  + Bool(var) turns it into boolean (True/False or 0 and 1)
  + Cannot put different types of variables in the same expression
  + Str(var) converts into string so converting numbers into strings you can then use them in the same expression
  + Python can automatically guess the type of data you are entering and it does not need to be stated explicitly
  + \ is the escape character
* Basic Python syntax
  + Operands are what you use in operations
  + Operators are the signs + - \* /
  + % will obtain the remainder of division e.g. 16 % 3 will have output of 1 (the remainder when dividing)
  + \*\* is to the power of
  + == means equals, so if x = 1, x == 1 checks whether it's true that x = 1, so the output will be boolean
  + != check if two values are not equal
  + On each side of the operand there could be expressions, values or variables
  + x = 1 and then x = 3, it will remain 3 and be overwritten
  + # comment with hash just like SQL
  + Indexing: "Friday"[4] will return the 5th character (because count starts from 0)
  + Everything written within one command follows an indentation, print is its own command and so cannot run within an indentation (within another command) UNLESS is it required for the command
  + Logical (boolean) operators: NOT, AND, OR
    - NOT first, then AND, then OR in an order
  + Identity operators: IS, IS NOT
    - IS is similar to ==
    - IS NOT is similar to !=
  + Conditional statements
    - if, if 5 == 15/3: print ("Hooray!")
    - If and ELSE statement is not specified, it will not output anything
    - else will provide the specified output outside of if
    - elif : else if, useful between an if and else statement
    - Can add as many elif statements as you like and always reads it top to bottom one command at a time (control flow)

Python functions

* To create a function, begin with "def" which is a keyword, then type a function name

|  |
| --- |
| def function\_name (parameters):  function body |

* + To call a function type its name and specify the parameter you want called

|  |
| --- |
| def plus\_ten (a):  return a + 10 |

* + Return returns the value from the function
  + So plus\_ten(2) will return 12
  + Return can be used only once in a function
  + Can also do it in this way:

|  |
| --- |
| def plus\_ten (a):  result = 10 + a  return result |

* + Print does not affect the calculation of the output and return does not visualise the output, it specified what a certain function is supposed to give back
  + They can be combined

|  |
| --- |
| def plus\_ten (a):  result = 10 + a  print ("Outcome")  return result |

* plus\_ten(2) will return: Outcome 12
* Can also have functions within functions and they can be specified in the def statement

|  |
| --- |
| def add\_five (a):  return a + 5  def multiply\_function (a):  return add\_five(a)\*3    multiply\_function(5) RETURNS 30 |

* Can combine conditional statements with functions

|  |
| --- |
| def compare\_the\_two(a,b):  if a > b:  print ("Greater")  if b > a:  print ("Less")  else:  print ("Equal")    compare\_the\_two(5,5) |

* Can specify multiple parameters, when calling can either say function\_name(a,b,c) or function\_name(a=1,b=2,c=3)
* Built-in functions: are already on the computer and can be applied directly
  + Type(): type of variable
  + Int(), float(), str(): transform the data type
  + Max(), min(): highest and lowest numbers of a sequence
  + Abs(): absolute value of the argument
  + Sum(): sum of all the elements in the list designated as an argument
  + Round(x,y): round number where x is the argument and y is how many decimal points
  + Pow(x,y): returns value x to the power of y (same as \*\*)
  + Len(): number of elements in an object, or its length
* Sequences
  + List: a type of sequence of data points
    - Contents must be placed within square brackets
    - To extract one part of the list, works as an index name\_of\_list[index\_element]
    - For the last position, can do -1, -2, -3 (not starting from 0)
    - Can assign new values as well name\_of\_list[3] = "new value"
    - To delete from a list: del name\_of\_list[3]
    - List values are treated as string
    - Len(list) will give the number of values in the list
    - Lists can also be made from other lists list1 = [list2, list3]
    - List.sort() sorts the list in alphabetical order
    - List.sort(reverse=True) to sort in descending
  + Methods
    - Same as function but they work slightly differently
    - object.method()
    - Dot operator (.) allows you to invoke a certain method
    - .append() will add a new value at the end of the list
    - .extend() will add another list to the existing list
    - object.extend(["Item1", "Item2"]
  + Slicing
    - Slice a list so that you only see a portion of it
    - List[1:2], the first will give the position of the 1st value we want and the last is the position of the value AFTER the last one that we want
    - Machine generated alternative text:
      participants — 
      participants 
      ['Oohn', ' 
      ' John 
      i la', 'haria•, 
      'lei la', 
      ' Maria' 
      ayne' , 
      'George' , 
      George , 
      Catherine' 
      out[2): 
      In 
      Out [3 
      In 
      'Catherine 
      cipantsL1:3J 
      'Leila', 't,taria'l 
    - Can leave the first number blank if you want the start of the list [:3] and leave blank at the end if want until the end of the list [3:]
    - Can also count from backwards so [-2:] will give everything from the last position to the end of the list
    - To find out the number of someone on a list do list\_name.index("name")
  + Tuples
    - A type of data sequence like a list but cannot be modified (cannot append or delete elements)
    - Tuples' list is placed within parentheses instead of square brackets
    - Tuple = (40, 20, 10)
    - It is a default sequence type so Tuple = 40, 20, 10 is also valid
    - Find index in the same way as list
    - Can split tuples: (age,years\_of\_school) = "30,17".split(',')
    - Can place tuples within lists and each tuple becomes a separate element within the list
    - Functions can provide tuples as return values (holding multiple values)

|  |
| --- |
| def Square\_info(x):  A = x\*\*2  P = 4\*x  print("Area and Permeter:")  return A,P |

* + - Square\_info(x) will return (9,12)
  + Dictionaries
    - Another type of storing data
    - Each value is associated with a key, each key and value form a key-value pair
    - Need curly brackets
    - Dict = {"k1":"cat", "k2":"dog", "k3":"mouse","k4":"fish"}
    - Value can be accessed by the key dict["k1"]
    - Can add new key-value pairs
    - Dict["k5"] = "horse"
    - Replacing the value is the same as well
    - Dict["k2"] = "parrot"
    - Can make lists within dictionaries
    - Dep\_workers = {"dep\_1": "Peter", "dep\_2":["Jennifer", "Michael", "Tommy"]}
    - Another way to create a dictionary:

|  |
| --- |
| Team = {}  Team["Point Guard"] = "Dirk"  Team["Shooting Guard"] = "Al"  Team ["Small Forward"] = "Sean"  Team["Center"]= "Hector" |

* + - Print(Team.get("Small Forward")) will return Sean
    - Print(Team.get("Coach")) will return None
* Iterations
  + Iteration is the ability to execute a certain code repeatedly
  + For loop: Repeats something a certain number of times, Python looks for each nth element and prints it

|  |
| --- |
| Even = [0, 2, 4, 6, 8, 10]  for n in Even:  print (n) |

* + - This will print all the elements with one on each line

|  |
| --- |
| Even = [0, 2, 4, 6, 8, 10]  for n in Even:  print (n, end = " ") |

* This will print all elements in a line separated by a space
* While loops: Repeats something until a certain condition is true or no longer true.
  + Can add an existing number into a loop which is called incrementing ( x = x + 2)

|  |
| --- |
| x = 0  while x <= 20:  print (x, end = " ") |

* + Can't stop here, this will create an infinite loop because x will always be smaller than 20

|  |
| --- |
| x = 0  while x <= 20:  print (x, end = " ")  x = x + 2 |

* This will produce: 0 2 4 … 20
* Can also write it as x+= 2 which means we are adding one increment of 2
* range() function
  + Can create lists that is a sequence of integers
  + range (start, stop, step)
  + Start is first value in the list, stop is the last number + 1, step is the increment
  + Stop is mandatory, if not specified start will be 0 and step will be 1
  + list(range(10)) will create a list from the range function: all integers from 0 - 9
* Conditional statements in loops

|  |
| --- |
| for n in range(10):  print (2\*\*n, end = " ") |

* + A name of a list does not need to be specified in a for loop, a range function is enough

|  |
| --- |
| for x in range(20):  if x % 2 == 0:  print (x, end = " ")  else:  print ("Odd", end = " ") |

* + Will print 0 Odd 2 Odd 4 Odd 6 Odd

|  |
| --- |
| x = [0, 1, 2]  for item in x:  print (item, end = " ") |
| for item in range(len(x)):  print (x[item], end = " ") |

* + Both will print 0 1 2 but the second way relies on the range(len(x)) = range(3) which is numbers from 0 to 2 and increment of 1 but relies on x being an index
* Rolling sum

|  |
| --- |
| def count(numbers):  total = 0  for x in numbers:  if x < 20:  Total +=1  return total |

* + What this does is that if there is a list, if the x is smaller than 20, the total will be incremented by 1, if it is more than 20 it will not grow
* Iterating over dictionaries

**\_Big\_O**

* Used to check how long it would take the computer to perform an algorithm
* Constant time: if algorithm is asked to look for the last item in an array, it doesn’t matter how many items there are because the length of time is independent from the number of items
* Linear time: the growth between the number of items (n) in an array and the time it takes to retrieve
  + Measured in O(n)
  + If the number of items doubles, it will double the time
* O(n^2): If there are now two tasks which need to be done as a loop, it will take O(n^2) time
  + Measured in O(n^2)
  + This happens in nested loops where the algorithm has to go through an item and then a nested loop
* Scaling: These are to be avoided at all costs, where the number scales based on a factorial
  + Measured in O(n!)
  + Can become exponentially large
* Consider the time complexity of the algorithm that you are using

**Stacks**

* Stack consists of elements that are stacked
* You can see the top item of the stack (peak) you can also remove the top item (pop), to add it back to the stack (push)
* Can reach a limit where you can't push more: stack overflow
* When you try to pop something that is not there: stack underflow

**Search and sort**

* Linear search
  + Search that starts with the first item on the list and goes through the list until it finds the item that you are looking for or until it runs out of items
  + As the list gets longer, takes longer to search through the time (linear time O(n))
* Binary search
  + Relies on the list that you are searching for to be sorted: like searching in a dictionary, wouldn’t search each page one by one, rather estimate first and the search from there
  + Goes to the middle item of the sorted list, if it's not the item you are looking for but is higher than the item you are looking for then the item must be in the left part of the list so you get rid of the right part of the list and repeat the process again and again until you reach the item you are looking for
* Insertion sort
  + Assume that the first item is a sorted list, then go to the next item and ask "is this item larger or smaller than the previous", if it is larger don’t do anything, if smaller then take the first item and shift it up one and then take the other item and put it in the first place (swap). These two now make their own sorted sub list and keep going until the end of the list