SECTION 6: METHODS AND FUNCTIONS, 2 hours 54 mins, 30 parts

- **27/29 Nested Statements and Scope**
  - **-> nested statements and scope**
    - how to write functions
    - how Python deals with the variable names we assign
      - **variable names are called in name space**
  - **-> in the .ipynb file**
    - x=25
    - def printer():
      - x = 50
      - return x <-**<u> if you call the function the value of x which is the global variable is printed (not the x which is stated in the local / definition of the function)</u>**
      - **<u>-> scope allows Python to have a set of rules to decide which variables we are referencing in the code -> LEGB - local enclosing global built-in <- this is the order in which values are called in the name space </u>**
      - -> this applies if e.g you have a function in a function
      - -> built-in are highlighted in certain colours
    - **-> local variables example**
      - in a lambda function
      - -> you need to list(the lambda function in here)
    - -> def greet():
      - name = 'Sammy'
      - def hello():
        - print('Hello ' +name)
      - hello()
    - greet()
    - **-> so**
      - -> we set the variable name equal to Sammy
      - -> then inside the hello function we are printing out the local variable name
      - -> the local variable is defined in the function definition
      - -> name is defined within that function
      - **-> she's commented out one of the lines of code and one of the variable names has been replaced with the next in LEGB -> the global value of that variable rather than the local is now being used (in the "global namespace")**
      - -> if x has multiple values in the JN -> then the one it uses when x is called is in order of LEGB
    - **-> another example**
      - **if you call a function with x as the argument of that function -> it doesn't change the global value of x in the JN -> this is scope (scope of the variable name in the function)**
      - **<u>-> you can declare in the definition of a function -> global x = 100</u> (then if you change it later in the function to a variable local to the function, the value of x used in the definition of the function is a local one - and everywhere else it's the global value of x (which was assigned this value in the definition of the function in this example)**
      - -> when you use the scripts again and again if the function definitions involve global

assignments to variables ->v this can cause errors if the function is used again and again