SECTION 5; PYTHON STATEMENTS, 1 hour 15 mins, 7 Parts

- **3/7 While Loops in Python**
  - -> code is executed **while the condition is true**
  - -> while some_boolean_condition:
    - #do something
  - -> else:

  - **-> In the .ipynb file**
    - x = 0
    - **while x < 5:**
      - **print(f'The current value of x is {x}) <- this is an f string literal**
      - x = x+1 <- the code is now looping and is running for as long as the condition is true
      - -> **you can get infinite while loops**
      - -> you can restart the kernel etc
      - **x +=1 <- this is the same as x=x+1**
    - else:
      - print("X is not less than 5")
    - **break, continue, pass for loops**
      - these are all Python keywords
      - **Break -> breaks out of the current closest enclosing loop**
      - **Continue: goes onto the top of the closest enclosing loop**
      - **Pass: does nothing at all**
      - x = [1,2,3]
      - for item in x:
        - **pass <- do nothing at all (this avoids syntax errors when you know it will output an error, it's this rather than just commenting it out**

    - mystring = 'Sammy'
      - for letter in mystring:
        - if letter == 'a':
          - continue <- this bypasses a **continue in loops bypasses whatever is being iterated through at that point in time**
        - print(letter) <- and it prints the letters **(you can iterate through entire strings)**
        - **-> replacing continue with break stops the loop entirely (when a comes up, it stops) -> it's like saying - stop if this comes up, or move onto the next thing if this comes up etc**

      - **-> another example**
        - x = 5
        - while x < 5:
          - print(x)
          - **x += 1 <- increasing the value of x by 1**
          - -> then printing its value for as long as it's <5
          - -> he's done an example where he's included a break statement and it's stopped iterating when x = 2
          - **-> control c interrupts the kernel**