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

- **4/7 Useful Operators in Python**
  - -> builtin functions and operators in Python
  - **-> in a .ipynb file**
    - **-> range**
      - mylist = [1,2,3]
      - for num in range(<- **SHIFT TAB TO SEE THE OPTIONS**
      - in this case, he's done
        - for num in range(3,10):
          - print(num)
          - -> **printing numbers in the range from 3 to 10**
          - -> **another example is range(0,11,2) <- it's start, stop, step**
        - **-> list(range(0,10,2)) <- this is an example of a generator (generating information rather than saving it to memory)**

    - **-> enumerate**
      - -> index_count = 0
      - **-> for letter in 'abcde':**
        - print('At index {} the letter is {}'.format(index_count, letter))
        - index count += 1
        - -> it's printing out the index of the letter and the letter in the string which its iterating through

      - **-> you can use the enumerate**
        - -> **enumerate is equivalent to using a dummy index**
        - -> for letter in word:
          - print(word[index_count])
          - index_count += 1
          - -> increasing the value of the index counter by 1
          - -> **when you iterate through the list, you can use print(word[index_count]) <- print the character at that index**
          - -> ENUMERATE:
            - word = 'abcde'
            - **for item in enumerate(word):**
              - print(index)
              - print(letter)
              - -> **if you ask it to print item -> it's e.g (1,'b')**
              - **-> you could alt. say -> for index, letter in enumerate(word):**
              - **-> this would allow you to access the index of the items you are iterating through**

      - **-> the zip function**
        - -> the opposite of enumerate
        - **-> this e.g zips two lists together**
          - mylist1 = [1,2,3]
          - mylist2 = ['a','b','c']
          - zip(mylist1,mylist2)
            - this returns that a zip has been created
          - **-> example using the zip function**
            - **-> for item in zip(mylist1,mylist2):**
              - **print(item)**

        - **-> this returns (1,'a') e.g <- a list of them**
        - **-> a list of tuples -> it's putting the two lists in tuples (~ coordinates)**
    - -> the length of the lists you're zipping together -> it will ignore the extra length if they don't all have the same length
    - **-> list(zip(mylist1, mylist2))**
    - **-> this returns an array of tuples**

- **-> to check if something is in a list**
    - **-> 'x' in [1,2,3]**
        - **-> returns False (x isn't in [1,2,3])**
        - -> instead of a list it can also be a string

    - **-> this also applies to dictionaries**
        - 'mykey'in {'mykey':345} <- this returns True
        - **d = {'mykey': 345}**
        - **345 in d.keys() <- is 345 in the keys**
        - **345 in d.values() <- is 345 in the values (you have the values and the keys)**

    - **-> mylist = [10,20,30,40,100]**
        - min(mylist)
        - max(mylist)
        - -> **min and max are keywords in Python -> don't call variables min and max**

- **-> random numbers**
    - **-> from random import shuffle <- then shift tab and can see the options from the documentation**
    - -> random shuffle -> randomly shuffles any list
    - **-> shuffle(mylist) <- it shuffles / scrambles the list - it's an in place function -> it's set the old list equal to the old list shuffled (it hasn't made any new lists)**
        - -> type(random_list) <- NoneType
    - mylist -> this is the same thing just shuffled
    - -> from random import randint
        - **randint(0,100) <- this returns a random integer in between 0 and 100**

- **-> the input function**
    - **result = input('Enter a number here: ')**
    - **-> running it asks the user to enter a number**
    - **-> then stores it in the variable result**
    - **-> input accepts what is entered into it as a string (e.g if you enter 30 into an input statement -> it's stored as '30' -> so you need to return int(result), or float(result))**