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

- **26/29 Lambda Expressions, Map, and Filter Functions**
  - **-> <u>lamda expressions map and filter</u>**
    - **<u>-> these are expressions to create anonymous (one time use) functions</u>**
    - **<u>-> like Einstein's lambda fudge</u>**

  - **-> in JN**
    - **map <- then shift tab for the documentation, it expects func and then *iterables (* is for any amount of them)**
    - -> def square(num):
      - return num**2
    - my__nums = [1,2,3,4,5]
      - we want to apply the square function to every item in the list -> **you could use the map function or a for loop**
      - **map(square,my_nums) <- use a function which works for a single number, but instead maps it to an entire array of values**

    - **-> another example**
      - **map is to - in this case, map a function which works on a single number to an entire array - it's saying - apply this function to an entire array of values**
      - -> def splicer(mystring):
        - if len(mystring)%2 ==0: <- then there are an even number of characters in the string
          - return even
        - else:
          - return mystring[0]
      - -> then she's testing the function on a list of strings
      - -> so we have a function which works on a single string
      - **-> then she uses map(name_of_function, array_name) <- and it runs the function on an entire array, without doing a loop**

    - **-> the filter function**
      - returns an iterator yielding the items of an iterable
      - -> def check_even(num):
        - return num%2 ==0
        - -> **she's defined a function which returns True of False**
      - -> then defined an array of numbers
      - **-> filter -> we have one function which takes a number and either returns true of false**
        - **list(filter(function_name, array_name))**
        - **-> this returns the elements in the list which return True when they are passed through the function**

    - **-> converting a function step by step into a lambda function**
      - -> def square(num):
        - result = num**2
        - return result
      - -> it's a function which squares the input
      - -> to turn it into a lambda expression / function

      - **-> she's simplified the function**

        - def square(num): return num**2
        - -> we don't give it a name
        - **-> lambda num: num**2 <- this is a lambda expression**
        - **-> with the 'map' -> this is where lambda expressions are useful**
        - **-> list(map(lambda num: num**2, mynums))**)
        - -> it's a similar idea with the filter function.
            - filter(filter(lambda num:num%2 ==0,mynums))

- **-> list(map(lambda x:x[::-1],names))**
    - -> not every function can be translated to a lambda expression
    - -> you need to be able to understand / read it when you come back to the code later