SECTION 12: PYTHON DECORATORS - 23 minutes, 2 parts

1/2 Decorators with Python Overview

- **decorators**
  - -> this allows us to decorate a function
  - -> how can we take a function and add something else to it
  - **-> an on / off switch -> were we can use certain parts of the code (or not)**
  - **-> this is the @ operator**
  - -> adding functions inside other functions

- **showing this concept without first using an @ operator**
  - **defining functions inside other functions, and then returning them**
    - -> he is defining one function inside another one
    - -> we are nesting one function in the definition of another
    - -> we have nested functions -> so calling one won't work depending on how the nesting was defined
    - -> we can have the larger function return the other functions which are defined inside of it
    - -> returning those entire functions
    - -> we have smaller functions defined in a larger one
    - -> we are having the larger function return the smaller ones which are inside of it
    - -> and we can put those different functions inside an if else block
    - -> depending on the condition, return the result of this nested function
    - -> we have a function defined inside another function

  - **we can also have functions as arguments to other functions**
    - -> in this example, he's defined a function
    - -> then used it as the input to another function
    - -> then returned its results
    - -> this is passing a function is an argument to another function
    - -> then returning the result to show that this is the case
    - -> we are creating an on / off switch when we get to a decorator

    - **-> he is doing another example**
      - -> we have a function defined inside another function
      - -> then we are executing the code outside of the original function
      - **-> he has defined a function inside a function**
      - **-> then returned the code from the indented function**
      - **-> it's a decoration -> it's a function inside a function which wraps around the code inside it**

    - **-> he then defines a decorator**
      - -> the wrapped function is being returned
      - -> he has defined a decorator using the @ syntax
      - -> and then defined a function
      - -> then called the name of the original function in another cell
      - -> this returns the output from the original function, with the extra output defined by the function
      - **-> if you don't want the decorator to be used, then you comment out the @ symbol**
      - **-> it's take the original function, and add this other function to it - but only if the @ on the second function isn't commented out**
      - **-> we can use this in Django / Flask for third party code**

- -> we are adding code onto other people's