**SECTION 14: ADVANCED PYTHON MODULES, 2 hours 23 minutes, 13 sections**
**2/13 Python Collections Module**

- **-> specialised container datatypes -> which are alternatives to Python default containers**
- -> counter
- **-> specialised tuple like containers**

- **-> in the .ipynb file**
  - **from collections import Counter**
  - she creates an array -> with repeats

  - **-> counting how many times 1 is repeated and how many times 2 is e.g**
    - you would create a for loop with a counter
    - -> **counter counts how many times one element in an array is repeated**
    - **-> Counter(array_name) <- this returns a dictionary with the amount of times one element is repeated -> this also works with lists**
      - **-> Counter is a dictionary subclass**
  - -> the keys are always the objects and the value is the count

  - **in an example**
    - **she's written a sentence**
    - **-> then Counter(sentence.lower().split()) <- this splits the sentence into an array where each element is one word in the list, then makes everything lowercase, then counts how many times each word in the list is repeated**
      - **-> you can . ....() . ...() <- use two methods at the same time**

  - **in another example**
    - -> **you can run counter on a string**
    - **-> if you type the variable_name then shift tab -> it shows you the different methods (functions) which are available**
    - **-> one of those options is string_name.most_common(3) -> this provides a list of tuples showing the most common ones**
    - **-> this can be used for NLP**
    - -> you can also turn a dictionary into a list

  - **the default dictionary**
    - **from collections import defaultdict**
    - **then she creates a dictionary**
      - **d = {'a':10} <- the key which is a and then the value**
      - **-> you can then d['a'] which returns 10**
        - -> if in this example you don't type the right key, it will return an error
        - -> **you can assign a default dictionary**
          - d = defaultdict() <- choosing the default value
          - **d = defaultdict(lambda: 0) <- this sets the default value equal to 0**
            - **-> when you put in a key which doesn't exist -> e.g d['WRONG KEY!'] -> the value returned is automatically 0**
            - -> rather than an error message
            - -> **setting a default value**

  - **the named tuple**
    - -> a default dictionary tries to return a default rather than a key error
    - -> the name tuple tries to expand on a normal tuple by expanding on named tuples

- in JN
  - **she defines a tuple (the syntax is the same as coordinates, a tuple is an immutable list)**
  - you can tuple_name[0] <- to return the name of the tuple
  - t**o make a named tuple (in between a class and a list but more convenient to define than a class and whose can be called like a list, but is immutable)**
    - **from collections import namedtuple**
    - Dog = namedtuple() <- then **shift tab to show the different options**

    - **Dog = namedtuple('Dog',['age','breed','name'])**
      - the key is Dog
        - -> these are like attributes in a class
        - **-> you can create instances of the Dog object**
          - -> sammy = Dog(age=5,breed='Husky',name='Sam')
          - -> type(sammy) is a __main__.Dog <- it's not the same as classes __init__(self, arguments)
          - -> you can call sammy.breed, sammy.age etc
          - -> sammy[0] is 5 (which is the age in this example) -> so you can either call it as if it were an attribute or in the form of a list
          - -> **these can be more convenient than classes**