**Programming Basics 3: Data Structures**

*Notes: Raveena Chhibber*, Emory University.

**Data Structures are exactly what they sound like! They are structures that are used to store related or relevant data.**

**The goal of this module is for you to learn about some of the four built-in data structures in python and the numpy array.** Specifically, we will learn about:

1. Common Structures
2. More on Built-in Structures
3. Numpy Arrays/N-Dimensional Arrays

**1. Common Structures**

The four built in types in Python are:

1. *list* (ex: l = [1,2,3 ] )
2. *tuple* (ex: t = (‘a’, ‘b’) )
3. *set* (ex: a = set([1, 2, 3, 4]) )
4. *dictionary* (ex: dict = ‘string’:’string value’, ‘A’: 95)

One additional type we will talk about today is the numpy array.

**Overall Note:** You can still type these variables using the type() function:

In[1]: t = ('a', 'b', '1', '2')

In[2]: type(t)

Out[2]: tuple

1. **A Short Digression – Objects and Classes**

**Principle:** Objects have their own functions. Examples:

* list.extend()
* dict.keys()

This is different syntax from built-in functions Examples:

* type(tuple)
* len(string)

**2. Built-in Python Structures**

**A) The List**

* Stores a *sequence* of items that can be strings, integers, floats and whatever else your heart desires.
* Separated by a comma and enclosed within brackets.
* Because you can add and remove items from this data type, it is considered *mutable.*

**To create a list use the syntax: l = [1, 2, 3, 4]**

For a list *l* some useful methods include:

1. *l.append(value)* – adds one value to the end of the list
2. *l.extend(value)* – adds a series of items to the end of the list
3. *l.insert(value, location)* – adds value to the list in a specific location
4. *l.remove(value)* – removes first item in list that has value of x.
5. *l.sort()* – sorts original list; modifies original list

Note: The len() method can consume most data types.

**Slicing**:

subjects = [“psychology”, “neuroscience”, “astrology”]

real\_sciences = subjects[0:2] **why not [0:1] ??**

**Practice:**

1. Create two lists (list1= 12, 79, 68; list2 = 45, 14, 0)
   1. What happens when you use .append to add values of list 2 to list 1?
   2. What could you have done instead?
2. Imagine you have a list of supplies you need for your lab that includes: pens, printer ink, snacks, scissors.
   1. Create this list. What type are the variables?
   2. Insert ‘duct tape’ onto the end.
   3. Sort the list.
   4. Return every other item in the list.
3. Challenge: Create lists of random numbers of length 4. Return True if the list contains the number 23. *(Hint: rand\_list = random.sample(range(1, 100), 4) generates a list of 4 random numbers between 1 and 100. You do not need to use loops for this but you can)*
4. **The Tuple**

* Pretty similar to lists.
* Usually stores data of different types
  + (Note: lists can be used to store data of different types it’s just sort of uncommon.)

To create a tuple, use the syntax: t = (‘17’, ‘y’, ‘5’, ‘m’)

This is different from a list: l = [‘17’, ‘y’, ‘5’, ‘m’]

For tuple *t* there are only two methods:

* 1. *t.count(value)* – returns the number of times you find value the tuple
  2. *t.index(value)* – returns first index of value

Note: To sort a tuple (or really anything that isn’t a list, use the sorted() function.

**Practice:**

1. Challenge: How do I make an empty tuple? What is type is x when I input x = (7)? When I input x = (7,)? When I input x= 7,?

1. **The Set**

* Sets are like lists but they contain unique items. Examples:
  + List of participants = [1, 5, 5, 9, 1]
  + set([1, 5, 5, 9, 1]) = 1, 5, 9,
* Unordered collections of objects – used most often when the presence of an object is more important than its order or frequency.
* Elements in a set **cannot** be duplicated, modified or indexed

For sets *days* and *months* some useful methods include:

1. participants.add(7) – adds additional value to the end of the set
2. participants.remove(9) – removes value from the set
3. **The Dictionary**

* Data structure where you can associate keys (e.g. grades) with their corresponding values (numerical ranges).
* Each **key** must be **unique** and must only be a **mutable** object (like a string or a tuple) but you can use mutable to immutable objects for the values in a dictionary.

To create a dictionary, use the syntax: d = {‘name’: ‘Arthur, quest: ’to seek the Holy Grail’, ‘Velocity of an unladen swallow’: ‘African or European’}

For dictionary *d* some useful methods include:

1. d.items() – returns key and value for all items in dictionary as tuples
2. d.keys() – returns keys for all items in the dictionary
3. d[‘key’] = ‘new value’ – updates value for key in dictionary
4. d[‘new key’] = ‘new value’ – adds new key-value pair to the dictionary.

Note: to combine dictionaries, use the .update function.

**Practice:**

1. Create dictionaries with the following data:

Participant 1: Participant 2: Participant 3:

Handed: Left Handed: Right Handed: Right

Score: 4.0 Score: 2.6 Score: 3.8

* 1. Return participant 1’s score
  2. Oh no! Your RA entered participant 1’s hand preference wrong; he or she is right handed. Return the new average score
  3. Challenge: Return the average score for participants that are right handed. *(Hint: you should create an additional data structure to hold all the dictionaries)*

**3. Numpy Arrays**

Python has special type of list known as the Numpy Array. It is typically faster, better at handling matrices of data, and commonly used by python libraries.

Lists: l = [1,2,3]

Numpy Arrays: arr = np.array([1,2,3])

*Check your understanding*: what allows us to type np not numpy?

Some useful variations include:

1. np.zeros((length, width), dtype) – creates an array of zeros of desired size
2. np.ones((length, width), dtype) – creates an array of ones of desired size

Once you’ve made your array *a,* some useful methods include:

1. a.shape – returns dimensions of array
   1. versus: a.size – returns number of array elements
2. a.dtype – returns data type of elements within array
3. s.reshape – reshapes/resizes the array
4. s.tranpose – transposes array dimensions

Note: all normal operators for addition, subtraction, multiplication etc. apply to arrays. Element-wise comparisons don’t necessarily need a for loop (e.g a == b will as if the values are equal for elements in the same position)

**Practice:**

1. Make a numpy array with the values ‘cat’, ‘dog’, and ‘ giraffe’.
   1. Get the shape of the array
   2. Get the size of the array
   3. Transpose the array
   4. Create an array of values ‘bird’, ‘monkey’ and ‘dog’
   5. Test if the two arrays are equal
2. Challenge: For the following arrays return a new array of the first and last values:

a = [1, 2, 3] → [1, 3]

b = [1, 2, 3, 4] → [1, 4]

c = [7, 4, 6, 2] → [7, 2]