Data Structures

Data structures refer to the collection or group of data in a particular structure.

Lists

Lists are the most commonly used data structure. Think of it as a sequence of data that is enclosed in square brackets and data are separated by a comma. Each of these data can be accessed by calling it's index value.

Lists are declared by just equating a variable to '[ ]' or list. We can use lists to hold an ordered sequence of values.

In [1]:
```
l = ['first', 'second', 'third']
print(l)
```

```
['first', 'second', 'third']
```

Lists can contain different types of variable, even in the same list.

In [2]:
```
another_list = ['first', 'second', 'third', 1, 2, 3]
print(another_list)
```

```
['first', 'second', 'third', 1, 2, 3]
```

```
In [3]: myList = [1,2,3,4]
        A = [myList]*3
        print(A)
        myList[2]=45
        print(A)

        [[1, 2, 3, 4], [1, 2, 3, 4], [1, 2, 3, 4]]
        [[1, 2, 45, 4], [1, 2, 45, 4], [1, 2, 45, 4]]
```

Lists are mutable; their contents can change as more statements are interpreted.

```
In [4]: l.append('fourth')
        print(l)

        ['first', 'second', 'third', 'fourth']
```

```
In [5]: list_one = [1, 2, 3]
        list_two = list_one
```

- The above code creates two different references (named `list_one` and `list_two`) to the *same* value `[1, 2, 3]`
- Because lists are mutable, changing them can have side-effects on other variables.
- If we append something to `list_one` what will happen to `list_two`?

```
In [6]: list_one.append(4)
        list_one

Out[6]: [1, 2, 3, 4]
```

```
In [7]: list_two

Out[7]: [1, 2, 3, 4]
```

List Methods

Method Name	Use	Explanation
append	alist.append(item)	Adds a new item to the end of a list
insert	alist.insert(i,item)	Inserts an item at the ith position in a list
pop	alist.pop()	Removes and returns the last item in a list
pop	alist.pop(i)	Removes and returns the ith item in a list
sort	alist.sort()	Modifies a list to be sorted
reverse	alist.reverse()	Modifies a list to be in reverse order
del	del alist[i]	Deletes the item in the ith position
index	alist.index(item)	Returns the index of the first occurrence of item
count	alist.count(item)	Returns the number of occurrences of item
remove	alist.remove(item)	Removes the first occurrence of item

State and identity

- The state referred to by a variable is *different* from its identity.
- To compare *state* use the `==` operator.
- To compare *identity* use the `is` operator.
- When we compare identity we check equality of references.
- When we compare state we check equality of values.

In [8]:
```
X = [1, 2]
Y = [1]
Y.append(2)
```

In [9]:
```
X == Y
```

Out[9]:
```
True
```

In [10]:
```
X is Y
```

Out[10]:
```
False
```

In [13]:
```
Y.append(3)
X
```

Out[13]:
```
[1, 2]
```

List Indexing

- Lists can be indexed using square brackets to retrieve the element stored in a particular position.
- Indexing is exactly similar to indexing in strings except that indexing in lists returns the entire item at that position whereas in strings, the character at that position is returned

In [14]:
```
print(l[0])
print(l[1])
```

```
first
second
```

List Slicing

Indexing was only limited to accessing a single element, Slicing on the other hand is accessing a sequence of data inside the list. In other words "slicing" the list.

Slicing is done by defining the index values of the first element and the last element from the parent list that is required in the sliced list. It is written as parentlist[a : b] where a,b are the index values from the parent list. If a or b is not defined then the index value is considered to be the first value for a if a is not defined and the last value for b when b is not defined.

In [ ]:
```
num_range = [0,1,2,3,4,5,6,7,8,9]
print(num_range[0:4])
print(num_range[4:])
```

Q. Given the list num_range, try printing:
a. Odd numbers
b. Even numbers
c. Numbers divisible by 3

Hint: Try using the step-size after the end slice index

In [ ]:
```
#Answers here
```

Q. Find the maximum, minimum and length from the list **num_range**.
Hint: Use max(), min() and len() respectively.

In [ ]:
```
#Answer here
```

In [ ]:
```
names = ['Earth','Air','Fire','Water']
```

Lets check if the items 'Air' and 'Wind' are present in the list or not

In [ ]:
```
'Wind' in names
```

In [ ]:
```
'Air' in names
```

In a list with elements as string, **max( )** and **min( )** is applicable. **max( )** would return a string element whose ASCII value is the highest and the lowest when **min( )** is used. Note that only the first index of each element is considered each time and if they value is the same then second index considered so on and so forth.

In [ ]:
```
mlist = ['bzaa','ds','nc','az','z','klm']
print(max(mlist))
print(min(mlist))
```

Here the first index of each element is considered and thus z has the highest ASCII value thus it is returned and minimum ASCII is a. But what if numbers are declared as strings?

In [ ]:
```
nlist = ['1','94','93','1000']
print(max(nlist))
print(min(nlist))
```

Even if the numbers are declared in a string the first index of each element is considered and the maximum and minimum values are returned accordingly.

But if you want to find the **max( )** string element based on the length of the string then another parameter 'key=len' is declared inside the **max( )** and **min( )** function.

In [ ]:
```
print(max(names, key=len))
print(min(names, key=len))
```

A string can be converted into a list by using the list() function.

In [ ]:
```
list('UpGrad')
```

List of words? Let's try using .split() which splits a string on the basis of spaces.

In [ ]:
```
test_string = 'I will now convert this to a list of words'
print(test_string.split())
```

What about splitting on something else? Like a period? and then try to make it like a normal sentence using .join?

In [ ]:
```
string_to_split = 'Hi.Can.we.get.rid.of.the.periods.here'
list_var = string_to_split.split('.')
print(list_var)
```

In [ ]:
```
#Let's join it back with spaces and reassign it to the original name
string_to_split = " ".join(list_var)
print(string_to_split)
```

In [ ]:
```
#Let's do it as a one-liner
string_to_split = 'Hi.Can.we.get.rid.of.the.periods.here'
out =  " ".join(string_to_split.split('.'))
print(out)
```

Q You have a list of email ids. The username for each email consists of all characters occurring before the @ symbol. You will need to write a code that will return a True or False after accepting user's input on the email id and check if the usernames is in the list of usernames or not. If it is present print True, else print False.
We have provided a template below. Fill it up and test it.

In [ ]:
```
list_of_usernames = []

#accept email id from user
input_name =

"""Treat the input to extract the username from email id - You might also want
to treat upper and lower case alike.
After extracting you might want to onvert it to lower case before looking it u
p in your list"""
input_username =

#print True if the username is present or print False. Do not use the if-else
 statement.
```

By the way, triple quotes are used for multiline comments. Don't you think something similar can be used during new user signups?

Sorting

In [ ]:
```
names.sort()
print(names)
names.sort(reverse=True)
print(names)
```

Reversing

In [ ]:
```
print(names)
print(list(reversed(names)))
```

Copying a list

Most of the new python programmers commit this mistake. Consider the following,

In [ ]:
```
lista= [2,1,4,3]
```

In [ ]:
```
listb = lista
print(listb)
```

Here, We have declared a list, lista = [2,1,4,3]. This list is copied to listb by assigning it's value and it get's copied as seen. Now we perform some random operations on lista.

In [ ]:
```
lista.pop()
print(lista)
lista.append(9)
print(lista)
```

In [ ]:
```
print(listb)
```

listb has also changed though no operation has been performed on it. This is because you have assigned the same memory space of lista to listb. So how do fix this?

If you recall, in slicing we had seen that parentlist[a:b] returns a list from parent list with start index a and end index b and if a and b is not mentioned then by default it considers the first and last element. We use the same concept here. By doing so, we are assigning the data of lista to listb as a variable.

In [ ]:
```
lista = [2,1,4,3]
listb = lista[:]
print(listb)
lista.pop()
print(lista)
lista.append(9)
print(lista)
```

In [ ]:
```
print(listb)
```

Take a look at the next block of code to get familiar with the list methods

In [ ]:
```
myList = [1024, 3, True, 6.5]
myList.append(False)
print(myList)
myList.insert(2,4.5)
print(myList)
print(myList.pop())
print(myList)
print(myList.pop(1))
print(myList)
myList.pop(2)
print(myList)
myList.sort()
print(myList)
myList.reverse()
print(myList)
print(myList.count(6.5))
print(myList.index(4.5))
myList.remove(6.5)
print(myList)
del myList[0]
print(myList)
```

In [ ]:
```
#Practice workspace
```

Tuples

- Tuples are another way to combine different values.
- The combined values can be of different types.
- Like lists, they have a well-defined ordering and can be indexed.
- To create a tuple in Python, use round brackets instead of square brackets

In [ ]:
```
first_tuple = (50, 'hello')
first_tuple
```

In [ ]:
```
print(first_tuple[0])
print(first_tuple[1])
print(type(first_tuple))
```

Tuples are immutable

- Unlike lists, tuples are *immutable*. Once we have created a tuple we cannot add values to it.

In [ ]:
```
first_tuple.append(25)
```

Tuples are similar to lists but only big difference is the elements inside a list can be changed but in tuple it cannot be changed. Think of tuples as something which has to be True for a particular something and cannot be True for no other values. For better understanding, let's use the **divmod()** function.

In [ ]:
```
xyz = divmod(10,3)
print(xyz)
print(type(xyz))
```

Here the quotient has to be 3 and the remainder has to be 1. These values cannot be changed whatsoever when 10 is divided by 3. Hence divmod returns these values in a tuple.

If you want to directly declare a tuple it can be done by using a comma at the end of the data.

In [ ]:
```
27,
```

27 when multiplied by 2 yields 54, But when multiplied with a tuple the data is repeated twice.

In [ ]:
```
2*(27,)
```

Values can be assigned while declaring a tuple. It takes a list as input and converts it into a tuple or it takes a string and converts it into a tuple.

In [ ]:
```
tup3 = tuple([1,2,3])
print(tup3)
tup4 = tuple('Hello')
print(tup4)
```

Slicing

Slicing is similar to lists

In [ ]:
```
print(tup3[1])
tup5 = tup4[:3]
print(tup5)
```

Built In Tuple functions

count() function counts the number of specified element that is present in the tuple.
index() function returns the index of the specified element. If the elements are more than one then the index of the first element of that specified element is returned

In [ ]:
```
example = ("Mumbai","Chennai","Delhi","Kolkatta","Mumbai","Bangalore")
print(example.count("Mumbai"))

print(example.index("Delhi"))
```

Dictionary

Dictionaries are more used like a database because here you can index a particular sequence with your user defined string. To define a dictionary, equate a variable to { } or dict()

In [ ]:
```
first_dict = {}
second_dict = dict()
print(type(first_dict), type(second_dict))
```

Dictionary Operators

Operator	Use	Explanation
[ ]	myDict[k]	Returns the value associated with k, otherwise its an error
in	key in adict	Returns True if key is in the dictionary, False otherwise
del	del adict[key]	Removes the entry from the dictionary

Dictionary works somewhat like a list but with an added capability of assigning it's own index style.

In [ ]:
```
first_dict['One'] = 1
first_dict['Twelve'] = 12
print(first_dict)
```

In [ ]:
```
'One' in first_dict
```

In [ ]:
```
del first_dict['One']
print(first_dict)
```

Dictionary Methods

Method	Use	Explanation
keys	adict.keys()	Returns the keys of the dictionary in a dict_keys object
values	adict.values()	Returns the values of the dictionary in a dict_values object
items	adict.items()	Returns the key-value pairs in a dict_items object
get	adict.get(k)	Returns the value associated with k, None otherwise
get	adict.get(k,alt)	Returns the value associated with k, alt otherwise

In [ ]:
```
capitals = {'Maharashtra':'Mumbai','Tamil Nadu':'Chennai'}

#get the capital of Maharashtra
print(capitals['Maharashtra'])

#create a new key Rajasthan whose capital(value) is Jaipur
capitals['Rajasthan']='Jaipur'
print(capitals)

#create another key-value pair
capitals['Punjab']='Chandigarh'
print(len(capitals))
```

Dictionary keys are unique and hence we cannot have two similar keys. Values can be the same though.

In [ ]:
```
capitals['Haryana'] = 'Chandigarh'
print(capitals)
```

values( ) function returns a list with all the assigned values in the dictionary./

In [ ]:
```
print(capitals.values())
print(type(capitals.values()))
```

In [ ]:
```
# to convert to a list use list()

capitals_list = list(capitals.values())
print(capitals_list)
print(type(capitals_list))
```

keys( ) function returns all the index or the keys to which contains the values that it was assigned to.

In [ ]:
```
print(capitals.keys())
print(type(capitals.keys()))
```

items( ) is returns a list containing both the list but each element in the dictionary is inside a tuple.

In [ ]:
```
print(capitals.items())

#convert to list of tuples
print(list(capitals.items()))
```

In [ ]:
```
#take a look at the block of code to understand
phoneext={'Sachin':1410,'Azhar':1137}
print(phoneext)

print(phoneext.keys())

print(list(phoneext.keys()))
print(phoneext.values())
print(list(phoneext.values()))

print(phoneext.items())

print(list(phoneext.items()))

print(phoneext.get("Dhoni"))

print(phoneext.get("Dhoni","Not a part of team"))
```

Sets

Sets are mainly used to eliminate repeated numbers in a sequence/list. It is also used to perform some standard set operations.

Sets are declared as set() which will initialize a empty set. Also set([sequence]) can be executed to declare a set with elements

In [ ]:
```
first_set = set()
print(type(first_set))
```

In [ ]:
```
second_set = set([1,2,2,3,3,4])
print(second_set)
```

Set Methods - Built in set functions

Method Name	Use	Explanation
union	aset.union(otherset)	Returns a new set with all elements from both sets
intersection	aset.intersection(otherset)	Returns a new set with only those elements common to both sets
difference	aset.difference(otherset)	Returns a new set with all items from first set not in second
issubset	aset.issubset(otherset)	Asks whether all elements of one set are in the other
add	aset.add(item)	Adds item to the set
remove	aset.remove(item)	Removes item from the set
pop	aset.pop()	Removes an arbitrary element from the set
clear	aset.clear()	Removes all elements from the set

In [ ]:
```
set_a = set([1,2,3])
set_b = set([2,3,4,5])
```

- Union: $X \cup Y$:
  union( ) function returns a set which contains all the elements of both the sets without repetition.

In [ ]:
```
set_a.union(set_b)
```

In [ ]:
```
#Union
X = {1, 2, 3}
Y = {4, 5, 6}
```

- Intersection: $X \cap Y$:
  intersection( ) function outputs a set which contains all the elements that are in both sets.

In [ ]:
```
set_a.intersection(set_b)
```

In [ ]:
```
#Intersection
X = {1, 2, 3, 4}
Y = {3, 4, 5}
```

- Difference $X - Y$:
  difference( ) function ouptuts a set which contains elements that are in set1 and not in set2.

In [ ]:
```
set_a.difference(set_b)
```

In [ ]:
```
#Difference
X = {1, 2, 3, 4}
Y = {3, 4, 5}
```

symmetric_difference( ) function ouputs a function which contains elements that are in one of the sets.

In [ ]:
```
set_b.symmetric_difference(set_a)
```

List to set and vice-versa

In [ ]:
```
list_with_duplicates = ['A','B','A','C','D','C','C','B']
print(set(list_with_duplicates))
```

In [ ]:
```
#Convert it to a list by wrapping it back to a list
print(list(set(list_with_duplicates)))
```