NPN Training

Training is the essence of success and we are committed to it.

PYTHON - Collections

Naveen P.N

Topics for the Module

After completing the module, you will be able to understand:

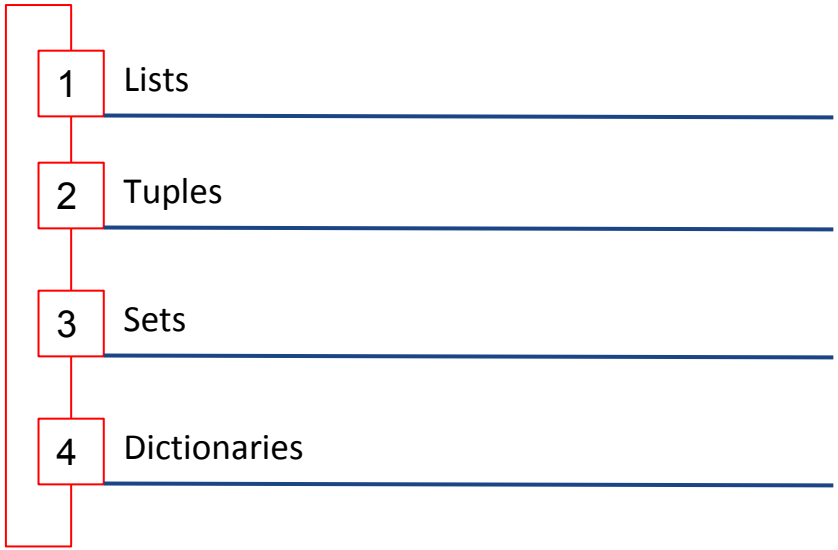
- Introduction to Python Collections
- Lists
- Tuples
- Sets
- Dictionaries
- Exercise [Hands on]

Introduction to Python Collections

A **collection** is similar to a basket that you can add and remove items from. In some cases, they are the same types of items, and in others they are different. Basically, it's a storage construct that allows you to collect things.

- For example, you might have a car type. You create several instances of the car and want some way to group all of those cars together and access them easily. This is the perfect scenario for a collection.
- The collection will survive in memory. You don't need to build the collection or create any type of scaffolding. All of that is provided for free.
- Just create a collection instance and start adding your cars. When you're ready, you can pull them out by name or by index (position within the collection).
- Python offers several built-in types that fall under a vague category called collections. While there isn't a formal type called **collection** in Python.

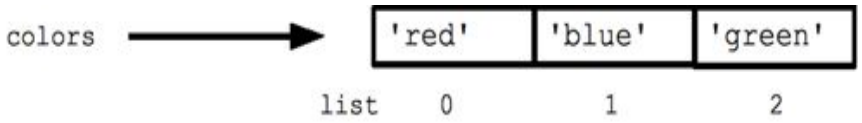
Types of Collections



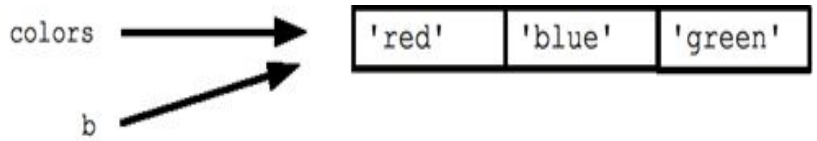
Lists

- A List in Python is defined as an ordered sequence of elements that can be dynamically altered .
  - Ordered means that each item in list has an index based on it's position in the list.
  - Sequence means that the elements are arranged in order,based on the indices.
  - "Dynamically altered" means more items can be added and existing items can be replaced or removed.

```
colors = ['red', 'blue',
'green']
  print colors[0]    ## red
  print colors[2]    ## green
  print len(colors)  ## 3
```




```
>>>b = colors
```



Assignment with an = on lists does not make a copy. Instead, assignment makes the two variables point to the one list in memory

Creating Lists

- In Python programming, a list is created by placing all the items (elements) inside a square bracket [ ], separated by commas.
- It can have any number of items and they may be of different types (integer, float, string etc.).

```
# empty list
my_list = []

# list of integers
my_list = [1, 2, 3]

# list with mixed datatypes
my_list = [1, "Hello", 3.4]

Also, a list can even have another list as an item. This is called nested list.
# nested list
my_list = ["mouse", [8, 4, 6], ['a']]
```

Accessing Lists

- There are various ways we can access the elements of the lists.

List Index:

- We can use the index operator [] to access an item in a list. Index starts from 0. So, a list having 5 elements will have index from 0 to 4.
- Trying to access an element other that this will raise an IndexError. The index must be an integer. We can't use float or other types, this will result into TypeError.
- Nested list are accessed using nested indexing.

```
>>> my_list = ['p','r','o','b','e']
>>> print(my_list[4])
e
>>> # Nested List
... n_list = ["Happy", [2,0,1,5]]
>>> print(n_list[0][1])
a
```

```
>>> print(my_list[-1])
e
>>> for i in my_list: print(i)
...
p
r
o
```

Searching Elements within Lists

- Check for existence
  - The in and not in operators help us to verify whether a particular element is present in a list or not .

```
>>> L=[5,2,3]
>>> 3 in L
True
>>> 4 not in L
True
```

- Counting occurence
  - The count() member function of list tells us how many instances of the specified object is present in the list .

```
>>> L=[5,2,3,2]
>>> L.count(3)
1
>>> L.count(2)
2
```

Searching Elements within Lists contd..

- Locating elements
  - The index() member function of list searches for the first occurrence of an element within a list and returns the index where it was found, and throws a ValueError if not found.

    Syntax : list.index(x[,i[,j]])

```
>>> L.index(2)
1
>>> L.index(2,3)
3
>>> L.index(2,0,2)
1
>>> L.index(53)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: 53 is not in list
```

Slicing, Inserting & Deleting elements in List

- A list slice is a sub-list extracted from an existing list. Syntax : list[start:end]
- Appending elements is to add one or more elements to the end of a list,the following function can be used. Syntax : list.append(x)
- Inserting and deleting elements is to add elements at any desired location within the list,we use the list.insert(i,x) function. Which inserts the element 'x' at index 'i' within the list. Syntax : list.insert(i,x)

```
>>> L=[5,2,3,2]
>>> L[1:3] # slicing of list
[2, 3]
>>> L[2:]=[]
>>> L
[5, 2]
>>> L1=[5,2,3]
>>> L2=[7,8,9]
>>> L1.extend(L2)
>>> L1
[5, 2, 3, 7, 8, 9]
```

```
>>> L.insert(50,9)
>>> L
[5, 2, 9]
>>> del L[1]
>>> L
[5, 9]
>>> L=[5,2,3,7,8,9]
>>> del L[1:5:2]
>>> L
[5, 3, 8, 9]
```

Remove, Pop & Clear functions on Lists

- If we want to delete a specific element regardless of it's position ,we can use the list.remove(x) which searches and removes the first occurrence of x in the list.
- Pop function allows us to delete an element at a specific position or at the end of a list and return the element that was deleted .
- Clear function deletes all the elements from a list and makes it empty .

```
>>> L.remove(3)
>>> L
[5, 8, 9]
>>> L.remove(3)
Traceback (most recent call last):
  File "<stdin>", line 1, in
<module>
ValueError: list.remove(x): x not in
list
```

```
>>> L=[5,2,3,2]
>>> L.pop(1)
2
>>> L
[5, 3, 2]

>>> L=[5,2,3,2]
>>> L.clear()
>>> L
[]
```

Copying, Adding & Multiplying Lists

- Adding the Lists:
  - Two lists can be concatenated to form a third list using the + operator

```
>>> L1=[1,3]
>>> L2=[7,8]
>>> L3=L1+L2
>>> L3
[1, 3, 7, 8]
```

- Multiplying Lists
  - A list L can be multiplied by an integer n to denote the list L repeated n times

```
>>> L=[5,2,3]*3
>>> L
[5, 2, 3, 5, 2, 3, 5, 2, 3]

#List * n is the same as n*List
```

Copying, Adding & Multiplying Lists contd..

- Assigning and Copying Lists
  - The = operator allows us to assign a list to a list variable.

```
>>> L1=[5,2,3]
>>> L2=L1
>>> L2
[5, 2, 3]

>>> L1=[5,2,3]
>>> L1[0]=9
>>> L1
[9, 2, 3]
```

Tuples

- A tuple in python is defined as an immutable ordered sequence of elements.
  - The contents of a tuple cannot be changed once created.
  - Ordered means that each item in a tuple has an index based on it's position in the tuple.

- Creating tuples from lists using tuple() :
  - A tuple can be created from a list and assigned to a variable using the tuple() function .

```
>>> T=tuple([5,2,3])
>>> T
(5, 2, 3)
```

- Creating Singleton Tuples : We cannot use the following syntax to create a Singleton tuple . T= (5)
- This is because the interpreter has no no way of knowing whether it has to treat (5) as an arithmetic expression or as a tuple.Here it assumes that we are dealing with an arithmetic expression.
- Hence we need to clearly indicate that we are dealing with a tuple, we need to use a comma with or without the parentheses. T=5, or T = (5,)

Accessing, Counting, Locating & Iterating Tuples

- Accessing the Tuples
  - We will be accessing individual elements of a tuple in exactly the same way as we do for lists

```
>>> T=(5,2,3)
>>> T[0]
5
>>> T[0]=9
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

- Counting Tuple elements
  - The len() function is used to count the number of elements in a tuple. The minimum size of any tuple is 0.

```
>>> T=(5,2,3)
>>> len(T)
3
```

Accessing, Counting, Locating & Iterating Tuples Contd..

- Iterating through tuple elements & Searching
  - Since ,the tuples are sequences,the for loop is directly compatible for iterating through the tuple.The in and not in operators will be used to verify whether a particular element is present in a tuple or not.

```
>>> T=(5,2,3)
>>> for i in T: print(i)
...
5
2
3
```

```
>>> 3in T
True
>>> 4 not in T
True
```

- Locating & Counting elements within tuples
  - The in and not in operators will be used to verify whether a particular element is present in a tuple or not.The count() function tells us how many instances of the specified object is present in the tuple .

```
>>> T.count(3)
1
>>> T.count(4)
0
```

```
>>> T.index(2)
1
>>> T.index(4)
throws error for this index
```

Accessing, Counting, Locating & Iterating Tuples Contd..

- Locating elements
  - The index() member function of tuple searches for the first occurrence of an element within a list and returns the index where it was found, and throws a ValueError if not found.

    Syntax : tuple.index(x[,i[,j]])

```
>>> T=(5,2,3,2)
>>>T.index(2)
1

>>> T.index(2)
Error
>>> L.index(2,0,2)
1
```

Locating Tuples Contd..

```
Form #2 :
Syntax : tuple.index(x,i)

>>>T=(5,2,3,2)
>>>T.index(2,2)
3

Form #3 :
Syntax : tuple.index(x,i,j)

>>>T=(5,2,3,2)
>>>T.index(3,0,2)
Error
```

Slicing, Adding & Multiplying Tuples .....

- A Tuple slice is a sub-tuple extracted from an existing Tuple. Syntax : tuple[start:end]
- The + operator is used for adding or concatenating the tuples.
- The * Operator is used for multiplying the tuples..

```
>>> T=(5,2,3,2)
>>> T[1:3] # slicing of tuple
[2, 3]
>>> T[1:]
(2,3,2)
>>>T[:3]
(5,2,3)
>>>T[:]
(5,2,3,2)
```

```
>>>T1=(1,3)
>>>T2=(7,8)
>>>T3=T1+T2
>>> T3
(1,3,7,8)

>>> T=(5,2,3)*3
>>>T
(5,2,3,5,2,3,5,2,3)
```

Assigning Tuples.....

- An element in a tuple can be a reference to some object .
- We cannot change the reference,But we can change the contents of the object.
- If a tuple contains a list,we can add/remove/change elements in the list for instance.
- The = operator allows us to assign a tuple variable .

```
>>> T1=(5,[],3)
>>> T2=T1
>>> T2
(5,[],3)
>>>T1[1].append(9)
>>>T1
(5,[9],3)
>>>T2
(5,[9],3)
```

```
>>> x = y = 5
>>> x
5
>>> y
5
>>>x,y = 2,3
>>>x
2
>>>y
3
```

Sets

- A Set is an unordered collection of unique elements .
  - Unordered means that elements in a set do not have an index by which they can be addressed ..
  - Duplicate elements are not allowed .
  - A set can contain only hashable elements.In Python immutable collections are hashable and mutable are not.

</>

```
>>> S=set([1,2,3,4,5])
>>> S
{1,2,3,4,5}

>>>S=set(range(1,6))
>>>S
{1,2,3,4,5}
>>>S={2,4,6,8}
>>>S
{8,2,4,6}
```

Counting and Iterating through Set elements ...

- The len() function is available to count the number of elements in a set .
- Since accessing particular set elements is not possible,iterating through a set is the only option to determine the contents of a set.

```
>>> S=set(range(1,6))
>>> S
{1,2,3,4,5}

>>>len(S)
5
>>>S=set()
>>>S
set()
>>>len(S)
0
```

```
>>> S=set(range(1,6))
>>> for i in S: print(i)

...
1
2
3
4
5
```

Adding Elements ...

- To Add individual elements to a set,the set.add(x) function will be used .
- To add a set of elements to another,the set.update(s) function is used.

```
>>> S1={2,4,6,8}
>>> S1
{8,2,4,6}
>>>S2={1,3,5,7}
>>>S2
{1,3,5,7}
>>>S1.update(S2)
>>>S1
{1,2,3,4,5,6,7,8}
```

```
>>> S=set(range(1,6))
>>> S
{1,2,3,4,5}
>>>S.add(9)
>>>s
{1,2,3,4,5,9}
```

Deleting Set elements ...

- The set.pop() function deletes an element from the set and returns the element deleted .We cannot predict which element will actually get deleted.Error if value not found.
- The set.remove(x) function removes the element x from the set .Error if value not found.
- The function set.clear() deletes all elements in the set.

```
>>> S=set(range(1,6))
>>> S
{1,2,3,4,5}
>>>S.pop()
1
>>>S
{2,3,4,5}
>>>S.pop()
2
```

```
>>> S=set(range(1,6))
>>> S
{1,2,3,4,5}
>>>S.remove(3)
>>>S
{1,2,4,5}
>>> S=set(range(1,6))
>>> S.clear()
>>>S
set()
```

Set Operations ...UNION

- The set.union(s) function returns a new set that represents the union of the set.
- The | operator plays the same role as set.union(s) .
- The set.update(s) function also finds the union,but stores it back into the original set.

```
>>> S1={1,2}
>>>S2={2,3}
>>> S1.union(S2)
{1,2,3}
>>>S1
{1,2}
>>> S1={1,2}
>>>S2={2,3}
>>> S1 | S2
{1,2}
```

```
>>> {1,2}.union({2,3})
{1,2,3}
>>>{1}.union({2},{3},{4})
{1,2,3,4}
>>>{1} | {2} | {3} | {4}
{1,2,3,4}
>>> S1={1,2}
>>>S2={2,3}
>>> S1.update(S2)
{1,2,3}
>>>S1 --->{1,2,3}
```

Set Operations ...INTERSECTION

- The set.intersection(s) function returns a new set that represents the intersection of the sets.
- The & operator plays the same role as set.intersection(s) .
- The set.intersection_update(s) function also finds the intersection,but stores it back into the original set.

```
>>> S1={1,2}
>>>S2={2,3}
>>> S1.intersection(S2)
{2}
>>>S1
{1,2}
>>> S1={1,2}
>>>S2={2,3}
>>> S1 & S2
{2}
```

```
>>> {1,2}.intersection({2,3})
{2}
>>>{1,2,3}.intersection({2,3,4},{1,
3,5})
{3}
>>>S1={1,2}
>>>S2={2,3}
>>>S1.intersection_update(S2)
>>>S1
{2}
```

Set Operations ...DIFFERENCE

- The set.difference(s) function returns a new set that represents the difference of the sets.
- The - operator plays the same role as set.difference(s) .
- The set.difference_update(s) function also finds the difference,but stores it back into the original set.

```
>>> S1={1,2}
>>>S2={2,3}
>>> S1.difference(S2)
{1}
>>>S1
{1,2}
>>> S1={1,2}
>>>S2={2,3}
>>> S1 - S2
{1}
```

```
>>> {1,2}.difference({2,3})
{1}
>>>{1,2,3}.difference({2,4,6},{3,6,
9})
{1}
>>>S1={1,2}
>>>S2={2,3}
>>>S1.difference_update(S2)
>>>S1
{1}
```

Dictionaries ….

- A dictionary is a collection of key-value pairs subject to the constraint that all the keys should be unique.
- The keys of a dictionary can be considered to be members of a set,with each key keeping track of a value.
- A dictionary object can be created and assigned to a variable using the dict() function .

```
>>> D=dict([('apple','red'),('grapes','green')])
>>>D
{'grapes': 'green' , 'apple': 'red'}

>>>D={'apple': 'red','grapes': 'green'}
>>>D
{'grapes': 'green' , 'apple': 'red'}
```

Accessing Dictionary elements ….

- A dictionary can be viewed as a special kind of array whose subscripts are strings instead of integers
- The [] operator is used to access an element of a list given it's key.
- An attempt to access a value using a key that does not exist in the dictionary results in a Key error.

```
>>>D={'apple': 'red','grapes': 'green'}
>>>D['apple']
'red'
>>>D['grapes']
'green'
>>>D={'apple': 'red','grapes': 'green'}
>>>D['grapes']='purple'
>>>D
{'grapes': 'purple' , 'apple': 'red'}
```

Iterating through Dictionary elements ….

- Since each element in a dictionary is a key-value pair,iterating through a dictionary is of multiple types.

```
Iterating through the keys of a dictionary :

>>>D={'apple': 'red','grapes': 'green'}
>>>for K in D: print(K)
…
grapes apple

Iterating through the values of a dictionary :

>>>D={'apple': 'red','grapes': 'green'}
>>>for K in D: print(D[K])
…
green red
```

Iterating through Dictionary elements contd ….

</>

```
Iterating through the key-value pairs of a dictionary :

>>>D={'apple': 'red','grapes': 'green'}
>>>for K,V in D.items():
        print(k,v)
…
…
Apple red
Grapes green
```

Adding and Deleting elements ….

- The elements can be added to a dictionary using the [] operator.
- The function dict.setdefault(key) returns the value associated with the key in the dictionary dict if the key exists and creates it with a value of None if the key does not exist.

</>

```
>>>D={‘apple’: ‘red’,‘grapes’: ‘green’}
>>>D[‘mango’]=’yellow’

>>>D={‘apple’: ‘red’,‘grapes’: ‘green’}
>>>D.setdefault(‘apple’)
‘Red’
>>>D.setdefault(‘pear’)
>>>D
```

Searching elements within dictionaries ….

- The in and not in operators are used to check if a key exists in a dictionary or not.
- The dict.get(key) is used to return the value corresponding to the key in the dictionary.It doesn't throw an error if we search for a key which doesn't exist.
- [] operator is also used to extract the value of a key.

```
>>>D={'apple': 'red','grapes': 'green'}
>>>'apple' in D
True
>>>D={'apple': 'red','grapes': 'green'}
>>>D.get('apple')
'Red'
>>>D.get('mango')
>>>D={'apple': 'red','grapes': 'green'}
>>>K='apple'
>>>V=D[K]
>>>print(V) --red
```

Deleting elements ….

- The del statement is used to delete an element from a dictionary,identifying using it's key.Error on non-existence.
- The dict.popitem() function removes an element from the dictionary and returns the pair in the form of a tuple.Generates an error when invoked on an empty dictionary .

</>

```
>>>D={'apple': 'red','grapes': 'green'}
>>>del D['apple']
>>>D
{'grapes': 'green'}

>>>D={'apple': 'red','grapes': 'green'}
>>>D.popitem()
('apple','red')
>>>D
{'grapes': 'green'}
```

Deleting elements contd ….

- The dict.pop(key) function deletes the element with the key from the dictionary and returns the corresponding value of the deleted key .Error on key non existing .
- The dict.clear() function clears out the dictionary,it removes all elements from the dictionary .

</>

```
>>>D={‘apple’: ‘red’,‘grapes’: ‘green’}
>>>D.pop(‘apple’)
‘red’
>>>D
{‘grapes’: ‘green’}

>>>D={‘apple’: ‘red’,‘grapes’: ‘green’}
>>>D.clear()
>>>D
{}
```

Key Takeaways

Hard work beats talent
when talent fails to **work hard**.