Python (Sequences)

Sequence in Python can be defined with a generic term as an ordered set which can be classified as two sequence types. They are mutable and immutable. There are different types of sequences in python. They are Lists, Tuples, Ranges.

Lists: Lists will come under mutable type in which data elements can be changed.

Tuples: Tuples are also like Lists which comes under immutable type which cannot be changed.

Ranges: Ranges is mostly used for looping operations and this will come under immutable type.

The common Sequence Operations are listed below:

x in s: Returns true if an item in s is equal to x

x not in s : Returns false if an item in s is equal to x

s+t : concatenation

s*n : adding s to itself n number of times

s[i] : ith item of s, index starts from zero

s[i:j]: slice of s from i to j

len(s) : length of s

max(s) : largest item in s

min(s) : smallest item of s

s.count(x): total number of occurrences of x in s

Python – Lists

Python Lists holds the data of any datatype like an array of elements and these are mutable means the possibility of changing the content or data in it. List can be created by giving the values that are separated by commas and enclosed in square brackets. Let us see different types of value assignments to a list.

Example:

```
List1=[10,20,30,40,50];
```

```
List2=['A','B','C','D'];

List3=[10.1,11.2,12.3];

List4=['html','java','oracle'];

List5=['html',10.1,10,'A'];
```

As we know the way strings can be accessed, same way Lists can be accessed. Below is example of indexing in python for your understanding again.

Example:

```
List1=[10,20,30,40,50];

    0  1  2  3  4  ---> Forward Indexing

   -5 -4 -3 -2 -1  ---> Backward Indexing
```

Accessing and slicing the elements in List

Now let us take a list which holds different datatypes and will access the elements in that list.

Example:


```
>>> list5=['html',10.1,10,'A'];

>>>list5[0]

'html'

>>>list5[1:2];

[10.1]

>>>list5[-2:-1];

[10]

>>>list5[:-1];

['html', 10.1, 10]

>>>list5[:-2];

['html', 10.1]

>>>list5[1:-2];
```

```
[10.1]
>>>list5[1:-1];
[10.1, 10]
>>>list5[-1];
'A'
>>>list5[3:];
['A']
```

Using Functions with Lists:

Example:

```
>>> list5=['html',10.1,10,'A'];
>>>len(list5)
4
>>> 10 in list5
True
>>> 'html' in list5
True
>>>num=[10,20,30,40];
>>>sum(num)
100
>>>max(num)
40
>>>min(num)
10
```

Checking if the Lists are mutable:

Example:

```
>>>score=[10,20,30,80,50]
>>>score
[10, 20, 30, 80, 50]
>>>score[3]=40
>>>score
[10, 20, 30, 40, 50]
```

List Comprehension:

List comprehension works like iterate operations as mentioned below.

Syntax:

```
[x for x in iterable]
```

Example:

```
>>>var=[x for x in range(5)];
>>>var
[0, 1, 2, 3, 4]
>>>var=[x+1 for x in range(5)];
>>>var
[1, 2, 3, 4, 5]
>>>var=[x for x in range(5) if x%3==0];
>>>var
[0, 3]
```

Adding Elements to a list:

We can add two lists as shown in the below example.

Example:

```
>>> var1=[10,20]
>>> var2=[30,40]
>>> var3=var1+var2
>>> var3
[10, 20, 30, 40]
```

Replicating elements in Lists:

we can replicate elements in lists as shown in the below example.

Example:

```
>>> var1*2
[10, 20, 10, 20]
>>> var1*3
[10, 20, 10, 20, 10, 20]
>>> var1*4
[10, 20, 10, 20, 10, 20, 10, 20]
```

Appending elements in Lists:

We can append an element to an existing list as shown in the below example.

Example:

```
>>>var1.append(30)
>>> var1
[10, 20, 30]
>>>var1.append(40)
>>> var1
```

```
[10, 20, 30, 40]

>>> var2

[30, 40]
```

Python – Tuples

Tuples are generally used to store the heterogeneous data which immutable. Even Tuple looks like Lists but Lists are mutable. To create a tuple, we need to use the comma which
separates the values enclosed parentheses.

Example:

```
>>> tup1=()       # Creating an empty tuple

>>> tup1

()

>>> tup1=(10)

>>> tup1

10

>>> tup1=(10,20,30);

>>> tup1

(10, 20, 30)

>>> tup1=tuple([1,1,2,2,3,3])

>>> tup1

(1, 1, 2, 2, 3, 3)

>>> tup1=("tuple")

>>> tup1

'tuple'
```

Using Functions with Tuples:

```
>>> tup1=(10,20,30);
>>>max(tup1)
30
>>>min(tup1)
10
>>>len(tup1)
3
```

Operators with Tuples:

```
>>> 20 in tup1
True
>>> 30 not in tup1
False
```

Slicing in Tuples:

```
>>>tup1[0:4]
(10, 20, 30)
>>>tup1[0:1]
(10,)
>>>tup1[0:2]
(10, 20)
```

Python – Dictionary

Dictionaries are created or indexed by key-value pairs. In which keys are immutable type.
Tuples can be used as keys, but lists cannot be used as keys. Just because lists are mutable.
Generally, the key-value pairs which are stored in the Dictionary can be accessed with the
key. we can delete a key value too. Let us see some examples.

Example:

```
>>>score={'maths':80,'physics':70,'chemistry':85}
>>>score
{'physics': 70, 'maths': 80, 'chemistry': 85}
>>>score['maths']
80
>>>del score['maths']
>>>score['maths']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'maths'
>>>score
{'physics': 70, 'chemistry': 85}


>>>score.keys()
dict_keys(['physics', 'chemistry'])


>>>keys=score.keys()
>>>keys
```

```
dict_keys(['physics', 'chemistry'])


>>>list(keys)

['physics', 'chemistry']
```

Python – Ranges

Range a kind of data type in python which is an immutable. Range will be used in for loops
for number of iterations. Range is a constructor which takes arguments and those must be
integers. Below is the syntax.

Syntax:

```
class range(stop)

class range(start, stop[, step])
```

stop: the value of stop parameter.

step: the value of step parameter. If the value is omitted, it defaults to 1.

start: the value of start parameter. If the value is omitted, it defaults to zero.

Examples:

```
>>>list(range(5))

[0, 1, 2, 3, 4]

>>>list(range(10,20))

[10, 11, 12, 13, 14, 15, 16, 17, 18, 19]

>>>list(range(10,20,5))

[10, 15]

>>>list(range(10,20,2))

[10, 12, 14, 16, 18]
```

```
# It will not take the float numbers
>>>list(range(0,0.1))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'float' object cannot be interpreted as an integer


>>>list(range(0,2))
[0, 1]


>>>list(range(0,1))
[0]


>>>list(range(0,10,5))
[0, 5]
```


Python – Sets

In this tutorial, we will learn about sets in python. A set is a datatype which holds an unordered collection with immutable and no duplicate elements. By the name, Set can be used for various mathematical operations. Mathematical operation may be union, intersection or difference, etc. Let us see the example of using the Set below.

Example:

```
>>> set1={'html','c','java','python','sql'}
>>>print(set1)
{'c', 'python', 'sql', 'html', 'java'}
# Below we have given duplicates
>>> set1={'html','c','java','python','sql','java'}
```

```
# we can observe that duplicates are ignored
>>>print(set1)
{'c', 'python', 'java', 'html', 'sql'}
>>> set1
{'c', 'python', 'java', 'html', 'sql'}
```

Membership testing in Sets:

```
>>> set1={'html','java','python','sql','java'}
>>> set1
{'python', 'java', 'html', 'sql'}
>>>print(set1)
{'python', 'java', 'html', 'sql'}
>>> 'c' in set1
False
>>> 'java' in set1
True
```

Sets Operations in Python:


```
>>> set1={'html','java','python','sql','java'}
>>> set2={'html','oracle','ruby'}

# Unique words in set1
>>> set1
{'python', 'java', 'html', 'sql'}
>>> set2
{'ruby', 'html', 'oracle'}
```

```
# words in set1 but not in set2

>>>set1-set2

{'python', 'java', 'sql'}


# Words in set1 or set2 or both

>>> set1 | set2

{'ruby', 'html', 'oracle', 'python', 'java', 'sql'}


# Words in both set1 and set2

>>> set1&set2

{'html'}


# Words in set1 or set2 but not both

>>> set1^set2

{'oracle', 'python', 'sql', 'ruby', 'java'}
```


Our Other Courses:

1. Artificial Intelligence
2. Data Science
3. Machine Learning
4. DevOps
5. AWS / Azure
6. IOT
7. Selenium
8. ETL Testing
9. Webservices Testing
10. Manual Testing
11. Blueprism
12. Uipath

For Any Queries Fell free to Call: 7730997544 (Teja)