Chapter-2 Revision Tour -II

Python Revision Tour –II

In this tutorial we will discuss the following topics

S.No.	Topics
1	Python: String
2	Python: List
3	Python: Tuple
4	Python: Dictionaries

Chapter-2 Revision Tour -II

Python: String

What is String?

A string is a *sequence* of *characters*. Strings are basically just a bunch of words.

Strings are *immutable*. It means that the contents of the string cannot be changed after it is created.

A literal/constant value to a string can be assigned using a single quotes, double quotes or triple quotes.

Declaring a string in python

>>>myString = "String Manipulation"

>>>mystring

Output: String Manipulation

Traversing a string:-

Traversing refers to iterating through the elements of a string, one character at a time. A string can be traversed using: for loop or while loop. For Example:

```
myname ="Amjad"
for ch in  myname:
    print (ch, end='-')
```

The above code will print:

A-m-j-a-d-

Access String with subscripts:

Let us understand with the help of an example:

myString = "PYTHON"

Chapter-2 Revision Tour -II

```
#Access String with subscripts

myString = "PYTHON"
#1) To access the first character of the string
print("1)To access the first character of the string: ",myString[0])
print ()

#2) To access the fourth character of the string
print("2)To access the fourth character of the string: ",myString[3])
print ()

#3)To access the last character of the string
print("3)To access the last character of the string: ",myString[-1])
print ()

#4)To access the third last character of the string
print("4))To access the third last character of the string: ",myString[-3])
print ()
```

OUTPUT:

```
=============== RESTART: D:/Amjad_CS/StringAccess_subscript.py
1)To access the first character of the string:  P

2)To access the fourth character of the string:  H

3)To access the last character of the string:  N

4))To access the third last character of the string:  H
```

Important points about accessing elements in the strings using subscripts:

Positive Index	0	1	2	3	4	5
myString	'P'	'Y'	'T'	'H'	'O'	'N'
Negative Index	-6	-5	-4	-3	-2	-1

- Positive subscript helps in accessing the string from the beginning
- Negative subscript helps in accessing the string from the end.
- Subscript 0 or negative n (where n is length of the string) displays the first element.

Example : A[0] or A[-6] will display 'P'

- Subscript 1 or –ve (n-1) displays the second element.

Chapter-2 Revision Tour -II

More on string Slicing:-

- The general form is: **strName [start : end]**
- **start** and **end** must both be integers
  - The substring begins at index **start**
  - The substring ends **before** index **end**
    - The letter at index **end** is not included

Let us understand with the help of an example:

Positive Index	0	1	2	3	4	5	6	7	8	9	10
String A	P	Y	t	h	o	n		P	r	o	g
Negative Index	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Lets we consider a string A= 'Python Prog'

S.No.	Example	OUTPUT	Explanation
1	print (A [1 : 3])	Yt	The print statement prints the substring starting from subscript 1 and ending at subscript 2
2	print (A [3 :])	hon Prog	Prints the substring starting from subscript 3 to till the end of the string
3	print (A [: 3])	Pyt	The print statement extract the substring before the third index starting from the beginning
4	print (A [:])	Python Prog	Omitting both the indices, directs the python interpreter to extract the entire string starting from 0 till the last index
5	print (A [-2 :])	og	For negative indices the python interpreter counts from the right side. So the last two letters are printed.
6	print (A[: -2])	Python Pr	It extracts the substring form the beginning. Since the negative index indicates slicing from the end of the string. So the entire string except the last two letters is printed.

Chapter-2 Revision Tour -II

String methods & built in functions:-

Lets Consider two Strings: str1="Save Earth" and str2='welcome'		
Syntax	**Description**	**Example**
len ( )	Return the length of the string	`>>> print (len(str1))` `10`
capitalize ( )	Return the exact copy of the string with the first letter in upper case	`>>>print (str2.capitalize())` `Welcome`

isalnum()	Returns True if the string contains only letters and digit. It returns False ,If the string contains any special character like _ , @,#,* etc.	`>>>print(str1.isalnum())`
		`FALSE`
		`The function returns False as space is an alphanumeric character.`
		`>>>print('Save1Earth'.isalnum())`
		`TRUE`

isalpha()	Returns True if the string contains only letters. Otherwise return False.	`>>> print('Click123'.isalpha())`
		`FALSE`
		`>>> print('python'.isalpha())`
		`TRUE`

isdigit()	Returns True if the string contains only numbers. Otherwise it returns False.	`>>>print (str2.isdigit())` `FALSE`
lower()	Returns the exact copy of the string with all the letters in lowercase.	`>>>print (str1.lower())` `save earth`
islower()	Returns True if the string is in lowercase.	`>>>print (str2.islower())` `TRUE`
isupper()	Returns True if the string is in uppercase.	`>>>print (str2.isupper())` `FALSE`
upper()	Returns the exact copy of the string with all letters in uppercase.	`>>>print (str2.upper())` `WELCOME`

Chapter-2 Revision Tour -II

Python: List

- The **list is a type of data** in Python used to **store multiple objects**.
- It is an **ordered and mutable collection** of comma-separated items between square brackets **[ ]**

Example:

myList = [10, None, True, "I am a student", 250, 0]

- Lists can be **indexed and updated**, **Examples:**

```
myList = [10, None, True, 'I am a student', 250, 0]
print(myList[3]) # outputs: I am a student
print(myList[-1])# outputs: 0
myList[1] = '?'
print(myList)    # outputs: [10, '?', True, 'I am a student, 250, 0]
myList.insert(0, "first")
myList.append("last")
print(myList)
# outputs: ['first', 10, '?', True, 'I am a student', 250, 0, 'last']
```

Here, index 1 (None) updated with value '?'

- **Lists can be nested**

Example:

```
myList = [1, 'a', ["list", 64, [0, 1], False]]
print(myList)
```

Output: `[1, 'a', ['list', 64, [0, 1], False]]`

Chapter-2 Revision Tour -II

- **List elements and lists can be deleted**

Example:

```
myList = [1, 2, 3, 4]
del myList[2]
print(myList)            # outputs: [1, 2, 4]
del myList               # deletes the whole list
```

- Lists can be **iterated** through using the ***for*** loop

Example:

```
myList = ["Rose", "Lily", "Lotus", "Dahlia", "Jasmine"]
for flower in myList:
        print(flower,end='-')

output: Rose-Lily-Lotus-Dahlia-Jasmine-
```

- The len() function may be used to **check the list's length**

Example:

```
myList = ["Rose", "Lily", "Lotus", "Dahlia", "Jasmine"]
print(len(myList))                      # outputs: 5
del myList[2]
print(len(myList))                      # outputs: 4
```

- If you have a list l1, then the following assignment: l2 = l1 does not make a copy of the l1 list, but makes the variables l1 and l2 **point to one and the same list in memory**.

Example:

```
Lst1= ['car', 'bicycle', 'motor']
print(Lst1)      # outputs: ['car', 'bicycle', 'motor']
Lst2 =Lst1
del Lst1[0]      # deletes 'car'
print(Lst2)      # outputs: ['bicycle', 'motor']
```

Chapter-2 Revision Tour -II

- If you want to copy a list or part of the list, you can do it by performing **slicing**:

Example

```
flowers = ['Rose', 'Lily', 'Lotus','Dahlia']

copyWholeflowers = flowers[:] # copy the whole list

print(copyWholeflowers) # output=['Rose', 'Lily', 'Lotus', 'Dahlia']

copyPartflowers = flowers[0:2] # copy part of the list

print(copyPartflowers)  #output=['Rose','Lily']
```

- You can use **negative indices** to perform slices, too.

Example:

```
sampleList = ["A", "B", "C", "D", "E"]

newList = sampleList[2:-1]

print(newList)          # outputs: ['C', 'D']
```

- The **start** and **end** parameters are **optional** when performing a slice:

Example:

```
myList = [1, 2, 3, 4, 5]
sliceOne = myList[2: ]
sliceTwo = myList[ :2]
sliceThree = myList[-2: ]
print(sliceOne)                 # outputs: [3, 4, 5]
print(sliceTwo)                 # outputs: [1, 2]
print(sliceThree)               # outputs: [4, 5]
```

Chapter-2 Revision Tour -II

- You can **delete slices** using the del instruction:

Example:

```
myList = [1, 2, 3, 4, 5]
del myList[0:2]
print(myList)        # outputs: [3, 4, 5]
del myList[:]
print(myList)        # deletes the list content

Outputs:   [ ]
```

- You can test if some items **exist in a list or not** using the keywords in and not in, **Example:**

```
myList = ["A", "B", 1, 2]
print("A" in myList)            # outputs: True
print("C" not in myList)        # outputs: True
print(2 not in myList)          # outputs: False
```

Chapter-2 Revision Tour -II

Python: Tuple

- **Tuples** are ordered and unchangeable (immutable) collections of data. They can be thought of as immutable lists. They are written in round brackets ( ):

Example:

```
myTuple = (1, 2, True, "a student", (3, 4), [5, 6], None)
print(myTuple)
```

OUTPUT: `(1, 2, True, 'a student', (3, 4), [5, 6], None)`

```
myList = [1, 2, True, "a tupleString", (3, 4), [5, 6], None]
print(myList)
```

OUTPUT: `[1, 2, True, 'a tupleString', (3, 4), [5, 6], None]`

Each tuple element may be of a different type (i.e., integers, strings, booleans, etc.).

What is more, tuples can contain other tuples or lists (and the other way round).

- **You can create an empty tuple like this:**

Example:

```
emptyTuple = ( )
print(type(emptyTuple))
```

OUTPUT: `<class 'tuple'>`

Chapter-2 Revision Tour -II

- **A one-element tuple may be created as follows:**

Example:

```
oneElemTup1 = ("two", )     # brackets and a comma

oneElemTup2 = "two",        # no brackets, just a comma
```

Note: If you remove the comma, you will tell Python interpreter to create a variable, not a tuple:

```
myTup1 = 1,
print(type(myTup1))
```

OUTPUT: `<class 'tuple'>`

```
myTup2 = 1
print(type(myTup2))
```

OUTPUT: `<class 'int'>`

- **You can access tuple elements by indexing them:**

Example:

```
myTuple = (1, 2.0, "tuplestring", [3, 4], (5, ), True)

print(myTuple[3])
```

OUTPUT: `[3, 4]`

Chapter-2 Revision Tour -II

- **Tuples are immutable, which means you cannot change their elements (you cannot append tuples, or modify, or remove tuple elements).**

The following snippet will cause an exception:

```
myTuple = (1, 2.0, "tuplestring", [3, 4], (5, ), True)
myTuple[2] = "guitar"
```

TypeError: 'tuple' object does not support item assignment

- **However, you can delete a tuple as a whole:**

```
myTuple = 1, 2, 3,
del myTuple
print(myTuple)
```

NameError: name 'myTuple' is not defined

- Tuple can be **iterated** through using the ***for*** loop

- **You can loop through a tuple elements**

```
t1 = (1, 2, 3)
for elem in t1:
    print(elem,end=' ')
```

OUTPUT: 1 2 3

Chapter-2 Revision Tour -II

- **check if a specific element is (not)present in a tuple**

```
t2 = (1, 2, 3, 4)
print(5 in t2)
print(5 not in t2)
```

OUTPUT: False

True

- **use the len() function to check how many elements there are in a tuple**

```
t3 = (1, 2, 3, 5)
print("Length of given tuple is: ",len(t3))
```

OUTPUT: Length of given tuple is: 4

- **or even join/multiply tuples**

```
t1 = (1, 2, 3)
t2 = (1, 2, 3, 4)
t3 = (1, 2, 3, 5)
t4 = t1 + t2
t5 = t3 * 2
print("join of t1 and t2 is: ",t4)
print("Multiplication of tuple t3 by 2: ",t5)
```

OUTPUT:

join of t1 and t2 is: (1, 2, 3, 1, 2, 3, 4)

Multiplication of tuple t3 by 2: (1, 2, 3, 5, 1, 2, 3, 5)

Chapter-2 Revision Tour -II

- **You can also create a tuple using a Python built-in function called tuple(). This is particularly useful when you want to convert a certain iterable (e.g., a list, range, string, etc.) to a tuple:**

```
myTup = tuple((1, 2, "computer"))
print(myTup)
```

OUTPUT: `(1, 2, 'computer')`

```
lst = [2, 4, 6]
print(lst)
```

OUTPUT: `[2, 4, 6]`

```
print(type(lst))
```

OUTPUT: `<class 'list'>`

```
tup = tuple(lst)
print(tup)
```

OUTPUT: `(2, 4, 6)`

```
print(type(tup))
```

OUTPUT: `<class 'tuple'>`

- **By the same fashion, when you want to convert an iterable to a list, you can use a Python built-in function called list():**

```
tup = 1, 2, 3,
lst = list(tup)
print(type(lst))
```

OUTPUT: `<class 'list'>`

Chapter-2 Revision Tour -II

Python: Dictionaries

- Dictionaries are unordered, changeable (mutable), and indexed collections of data. (**Note:** In Python 3.6x dictionaries have become ordered by default).
- Each dictionary is a set of ***key: value*** pairs.
- You can create it by using the following **Syntax:**

```
my_dict = {'key1': 'value1','key2': 'value2','key3': 'value3'...'keyn': 'valuen'}
```

Dictionary Keys

- A dictionary as an unordered set of ***key: value*** pairs
- Dictionary keys must be ***unique***
  - A key in a dictionary is like an index in a list
  - Python must know exactly which value you want
- Keys can be of any data type
  - As long as it is ***immutable***

Dictionary Values

- Dictionary keys have many rules, but the values do not have many restrictions
- They do not have to be unique
- They can be mutable or immutable

To Create Empty Dictionary:

- The function dict ( ) is used to create a new dictionary with no items. This function is called built-in function. OR
- We can also create dictionary using { }.

Chapter-2 Revision Tour -II

Example:

```
D=dict()
print (D)

d={}
print(d)
```

OUTPUT: { }

- ***To create a dictionary***, use curly braces, and a colon (:) to separate keys from their value

Example:

```
computer={'input':'keybord','output':'monitor',
          'language':'python','os':'windows- 8',}
print (computer)
```

OUTPUT: {'input': 'keybord', 'output': 'monitor', 'language': 'python', 'os': 'windows- 8'}

In the above example

input, output, language and os are the keys of dictionary

keyboard, monitor, python and windows- 8 are values.

- If you want ***to access a dictionary item***, you can do so by making a reference to its key inside a pair of square brackets.

Example:

```
compdic={'input':'keybord','output':'monitor',
         'language':'python','os':'windows- 8',}

item1 = compdic["output"]
print(item1)
```

OUTPUT: monitor

Chapter-2 Revision Tour -II

- By using the ***get()*** method

Example:

```
compDict={'input':'keybord','output':'monitor',
          'language':'python','os':'windows- 8',}
item2 = compDict.get("language")
print(item2)
```

OUTPUT: python

- If you want to ***change the value associated with a specific key***, you can do so by referring to the item's key name in the following way:

Example:

```
compDict={'input':'keybord','output':'monitor',
          'language':'python','os':'windows- 8',}

compDict["output"] = "printer"
item = compDict["output"]
print(item)
```

In this example we change the value associated with output key to printer (i.e. change monitor to printer)

OUTPUT: printer

- To ***add or remove a key*** (and the associated value), use the following syntax:

Example:

```
compDict={'input':'keybord','output':'monitor',
          'language':'python','os':'windows- 8',}


compDict["memory"] = 'HDD' # create/add a key-value pair
print("Dictionary after adding new value",compDict)

del compDict["memory"]     # detele the key memory with value
print("Dictionary after deletion",compDict)
```

Chapter-2 Revision Tour -II

OUTPUT: Dictionary after adding new value {'input': 'keybord', 'output': 'monitor', 'language': 'python', 'os': 'windows- 8', 'memory': 'HDD'}

Dictionary after deletion {'input': 'keybord', 'output': 'monitor', 'language': 'python', 'os': 'windows- 8'}

- You can also ***insert an item*** to a dictionary by using the ***update()*** method, and ***remove the last element*** by using the ***popitem()*** method

Example:

```
compDict={'input':'keybord','output':'monitor',
          'language':'python','os':'windows- 8',}

compDict.update({"network" : "wi-fi"})
print(compDict)

compDict.popitem()
print(compDict)
```

update() method add new pair-value(i.e. 'network':' wi – fi') to the dictionary and

popitem() remove the last element from the dictionary.

OUTPUT: {'input': 'keybord', 'output': 'monitor', 'language': 'python', 'os': 'windows- 8', 'network': 'wi-fi'}

{'input': 'keybord', 'output': 'monitor', 'language': 'python', 'os': 'windows- 8'}

- You can use the ***for*** loop to loop through a dictionary

Example:

```
compDict={'input':'keybord','output':'monitor',
          'language':'python','os':'windows- 8',}

for item in compDict:
    print(item)
```

OUTPUT: input

output

language

os

Chapter-2 Revision Tour -II

- If you want to loop through a dictionary's keys and values, you can use the ***items()*** method

Example:

```
compDict={'input':'keybord','output':'monitor',
          'language':'python','os':'windows- 8',}
for key, value in compDict.items():

    print(key, ":", value)
```

OUTPUT: input : keybord
output : monitor
language : python
os : windows- 8

- To check if a given key exists in a dictionary, you can use the ***in*** keyword:

Example:

```
compDict={'input':'keybord','output':'monitor',
          'language':'python','os':'windows- 8',}
if "os" in compDict:
    print("Yes")
else:
    print("No")
```

OUTPUT: Yes

Chapter-2 Revision Tour -II

- You can use the ***del*** keyword to remove a specific item, or delete a dictionary.
- To remove all the dictionary's items, you need to use the ***clear()*** method:

Example:

```
compDict={'input':'keybord','output':'monitor',
          'language':'python','os':'windows- 8',}

print(len(compDict))    # outputs: 4
del compDict["os"]      # remove an item
print(len(compDict))    # outputs: 3

compDict.clear()        # removes all the items
print(len(compDict))    # outputs: 0

del compDict            # removes the dictionary
```

- To copy a dictionary, use the ***copy()*** method:

Example:

```
compDict={'input':'keybord','output':'monitor',
          'language':'python','os':'windows- 8',}
copyDict = compDict.copy()

print("Copied Dictionary :",copyDict)
```

OUTPUT: `Copied Dictionary: {'input': 'keybord', 'output': 'monitor', 'language': 'python', 'os': 'windows- 8'}`