PYTHON PROGRAMMING

UNIT-IV: Tuples: Creating tuples-accessing values-tuple assignment-tuples as return values- variable length argument tuples-basic tuple operations-built-in tuple functions. Dictionaries: Creating and accessing values in a dictionary - updating and deleting elements -operations and built-in dictionary methods. Files: Opening and Closing a file-Reading and Writing a file.

TEXT BOOK

1. E.Balagurusamy, "Introduction to Computing and Problem Solving Using Python", McGraw Hill Education Private Limited, 1st Edition, New Delhi.

TUPLES

- Tuples are series of values of different types separated by commas(,)
- Assessed by index values, starting from 0.
- The values in tuples cannot be replaced with another, once tuples are created.

I Creating Tuples

- All items are placed inside parenthesis separated by commas and assigned to a variable.
- Tuples can have any number of different data items(int, float, string, list, etc)

Ex:

a. Tuple with integer data item

```
T=(4,2,9,1)
Print t
```

Output

```
(4,2,9,1)
```

b. Tuple with different data types

input

```
Tm=(2,30,’python’,5.8)
Print tm
```

Output

```
(2,30,’python’, 5.8)
```

c. Nested Tuple

Input

```
nt=(“python”,[1,4,2],[“sita”,3.9])
Print nt
```

Output

```
(“python”,[1,4,2],[‘sita’,3.9])
```

d. Tuple can be created without parenthesis

input

```
T=4.9,6,’house’
Print T
```

Output

```
(4.9,6,’house’)
```

Note: creating a tuple with one element we need to add a final comma after the item or element in order to complete the assignment of the tuple.

Ex:

```
t=(“house”,)
Type(t)
```

Output

```
<type ‘tuple’>
```

II Accessing values in tuple:

Use the index number enclosed in square brackets along with the name of the tuple.

Ex: using square bracket

```
T1=('physics', 'chemistry', 'maths')
T2=(10,20,30)
Print t1[1]
Print t2[2]
```

Output

```
Chemistry
30
```

Ex: using slicing

```
T1=('physics', 'chemistry','maths')
T2=(10,20,30)
T2[1:2]
T1[:1]
```

Output

```
20
'physics'
```

III Tuples are immutable

The values or items in the tuple cannot be changed, once it is declared if we want to change the values we have to create a new tuple.

Ex:

```
T1=[12,15,'python',2.3]
T1[2]="hello"
```

Output

```
Type error:'tuple' object does not support items assignment.
```

IV Tuple Assignment

- Assignment of values to a tuple of variables on the left side of the assignment from the tuple of values on the right side of the assignment
- The number of items on left side must match the number of elements on the right side of the assignment

Ex:

```
#creating the tuple
A=(‘221’, ’sony’, ’ragul’, 1971)
#tuple assignment
(id,name,name1,year)=A
Print id
Print name1
```

Output

```
221
ragul
```

Swapping the values of two variables

Ex:
Temp=x
X=y
Y=temp
	Ex:
	X=3
	Y=4
	x, y=y,x
	Print x
	Print y
	Output
	4
	3

V tuples as return values

Function returns only one value by returning tuple, a function can return more than one value.

#Function definition	#Function calling

Def div(a,b):
Q=a/b
R=a%b
Return q,r
	X=10
	Y=3
	T=div(x,y)
	Print t
	Output
	(3,1)
	Type(t)
	Output
	<type ‘tuple’>

Def max_min(t):
Return max(t),min(t)
	A=[10,3,2,100]
	Max_min(a)
	Output
	(100,2)

VI variable length argument tuples:

A variable name that is preceded by an astrisk(*) collects the arguments into a tuple.

#function definition	#Function calling

Def traverse(*t):
# *t can take any number of arguments 	
i=0
While i<len(t)
Print t[i]
i=i+1
	Traverse(1,2,3,4,5)
	Output
	1
	2
	3
	4
	5

Ex: an email address is provided hello@ python.org using tuple assignment, split the username and domain from the email address	A=hello@python.org Usname,domain=a.split('@') Print usname Print domain Output Hello Pyhton.org

Write a function called sumall that takes any number of arguments and returns their sum	Write a function called circle info which takes the radius of circle as arguments and returns the area and circumference of the circle

#function definition
Def sumall(*t)
i=0
Sum=0
While i<len(t):
Sum=sum+t[i]
i=i+1
Return sum
	#function definition
	Def circleinfo(r):
	C=2*3.14*r
	A=3.14*r*r
	Return(c,a)
	#function call
	Circleinfo(10)
	output
	(62,83,314.15)

VII basic tuple operations

1. concatenation: + operator concatenates two tuples

Ex:

```
t1=(1,2,3)
t2=(4,5,6)
t3=t1+t2
Print t3
```

Output

```
(1,2,3,4,5,6)
```

2. Repetition: it repeats the tuples a given number of times * operator is used.

Ex:

```
T1=('ok')
t1*5
```

Output

```
('ok',' ok',' ok', 'ok', 'ok')
```

3. In operator

It tells the user that the given element exists in the tuple or not. It gives the boolean output true or false.

Ex:

T1=(10,20,30,40)

20 in t1

50 in t1

Output

True

false

Ex:

T1=(‘rose’,’jasmine’,’lilly’)

Jasmine in t1

Output

true

4. Iteration-iteration can be done in tuples using for loop. It helps in traversing the tuple.

Ex:

```
T1=(1,2,3,4,5)
For x in t1:
Print x
```

Output

```
1
2
3
4
5
```

BUILT-IN TUPLE FUNCTIONS

FUNCTIONS	DESCRIPTION
Cmp(tuple1,tuple2)	It compares the items of two tuples
Len(tuple)	It returns the length of a tuple
Zip(tuple1,tuple2)	It ‘zips’ elements from 2 tuples into a list of tuples
Max(tuple)	It returns the largest value among the elements in a tuple
Min(tuple)	It returns the smallest value among the element in a tuple
Tuple(seq)	It converts a list into a tuple

Ex:

```
T1=(‘physics’, ’ chemistry’, ’mathematics’)
T2=(10,20,30,40,50)
Len(t1)          #output 3
Len(t2)          #output 5
Zip(t1,t2)        #output [(‘physics’,10),(chemistry’,20),(‘mathematics’,30)]
Max(t1)          #output ‘physics’   #outputs in alphabetical order
Max(t2)          #output 50
Min(t1)           #’chemistry’
Min(t2)          # 10
```

Ex:

S=(‘Hello’)

T=(‘python’)

Zip(s,t)

Output:

[(‘H’, ‘p’),(‘e’,’y’),(‘l’,’t’),(‘l’,’h’),(‘o’,’o)]

Note: if the length of tuples are not same then the resulting tuple after applying the zip function will have the length of shorter tuple.

DICTIONARIES

- Python dictionary is an unordered collection of items or elements.
- Dictionary is said to be mapping between some set of keys and values. The mapping of a key and value is called key-value pair and together they are called one item or element.
- Key and value is separated by(:). The items or elements in a dictionary are separated by commas and all the elements must be enclosed in curly braces.

- A pair of curly braces with no values in between is known as empty dictionary.
- The values in a dictionary can be duplicated, but the keys in the dictionary are unique.

1. Creating a Dictionary

Values in a dictionary→ can be any data type

Keys in a dictionary → string, number or tuple

Ex: empty dictionary

D1={}

Print d1

Output

{}

Dictionary with integer keys

D1={1:’red’, 2: ‘yellow’ , 3: ‘green’}

Print d1

Output ={1:’red’, 2: ‘yellow’ , 3: ‘green’}

Dictionary with mixed keys

Ex:

D1={‘name’:’Radha’, 3:[‘Hello’,2]}

Print d1

Output

{3:[‘hello’,2],’name’:’radha’}

Note: dictionary has internal mechanism to sort the keys and then print them.

Note: python provides a built-in functions dict() for creating a dictionary.

Ex:

D1=dict({1:’red’, 2:’yellow’})

D2=dict({1,’red’,2,’yellow’})

D3=dict(one=1,two=2,three=3)

Print d3

output

{‘three’:3, ‘two’:2, ‘one’:1}

2. Accessing values in a dictionary

- We can use the value of the key enclosed in square bracket
- We can use get() method

Ex:

```
D1={‘name’:’radha’, ‘age’:30}
D1[name]            #output ‘radha’
Print d1[‘name’]    #output radha
Print d1[‘age’]     #output 30
D1.get(‘name’)      #output radha
D1.get(‘age’)       #output 30
```

Note: when we try to access a key that does not exists in a dictionary, an error occurs.

3. Updating Dictionary

- Dictionaries are mutable
- Values in a dictionary can be changed, added or deleted

Ex:

D1={‘name’:’Radha’, ‘age’:30}

D1[‘age’]=35

Print d1 #output {‘age’:35, ‘name’:’Radha’}

D1[‘address’]=‘coimbatore’

Print d1 #output {‘age’:35, ‘name’:’radha’, ‘address’: ‘coimbatore’}

4. Deleting elements from dictionary

- Removed or deleted using pop() method
- Popitem() is used to remove or delete and return an arbitary item from the dictionary
- Clear() method removes all the items/elements from the dictionary at once. Now the dictionary becomes empty dictionary.

Ex:

```
D1=(1:1, 2:8, 3:9, 4:64, 5:125, 6:216)
D1.pop(3)        #remove a particular item              #output 9
D1.popitem()     #remove arbitary item(first item)      #output (1,1)
D1.popitem()     #output (2,8)
D1               #output (4:64,5:125,6:216)
Del d1[6]
D1               #output (4:64, 5:125)
D1.clear()
D1               #output {}
Del d1           #output traceback most recent call last
```

Properties of dictionary keys:

Keys have restrictions while defining them:

1. One key in a dictionary cannot have two values
   - i.e. duplicate keys are not allowed in a dictionary, they must be unique

   Ex:

   D1={‘name’:’radha’, age:30, ‘name’:’kavitha’}

   Print d1[‘name’]

   Output

   Kavitha

2. Keys are immutable

- We can use string, integers or tuples for dictionary keys.
- [‘key’] →key within bracket is not allowed.

Ex:

```
D1={[‘name’]:’radha’, ‘age’:30}
```

Output

```
Traceback most recent call last
```

Operations in dictionary

1. Traversing: Traversing is done in on the basis of keys. For loop is used, which iterates over the keys and prints the corresponding values.

Ex:

```
Def d1(d):
For c in d:
    print c,d[c]
d2={1:’a’ , 2:’b’, 3:’c’}
d1(d2)
```

Output

```
1 a
2 b
3 c
```

2.Membership

The membership operator i. in ii. Not in

We can test whether the key is in dictionary or not.

If the key found $\longrightarrow$ True

If key not found $\longrightarrow$ false

Ex:

```
C={1:1, 2:8, 3:27}
3 in c          #output true
7 not in c      #output true
10 in c         #output false
```

Built-in Dictionary methods

all(dict)	**It is a boolean type function which returns true if all keys of dictionary are true or dictionary is empty.**
any(dict)	Returns true, if any key of the dictionary is true
len(dict)	It returns length in the dictionary(items)
sorted(dict)	It returns the sorted list of keys
str(dict)	It produces the printable string representation of the dictionary

Ex:

```
C={1:1, 2:8, 3:27, 4:64, 5:125}
all(c)          # true
any(c)          # true
len(c)         # 5
sorted(c)    #[1,2,3,4,5]
str(c)          #'{1:1, 2:8, 3;27, 4:64, 5:125}'
```

Reading and Writing to text files in Python

Python provides inbuilt functions for creating, writing and reading files. There are two types of files that can be handled in python, normal text files and binary files (written in binary language,0s and 1s).

- **Text files:** In this type of file, Each line of text is terminated with a special character called EOL (End of Line), which is the new line character ('\n') in python by default.
- **Binary files:** In this type of file, there is no terminator for a line and the data is stored after converting it into machine understandable binary language.

File Access Modes

Access modes govern the type of operations possible in the opened file. It refers to how the file will be used once its opened. These modes also define the location of the **File Handle** in the file. File handle is like a cursor, which defines from where the data has to be read or written in the file. There are 6 access modes in python.

- **Read Only (‘r’) :** Open text file for reading. The handle is positioned at the beginning of the file. If the file does not exists, raises I/O error. This is also the default mode in which file is opened.
- **Read and Write (‘r+’) :** Open the file for reading and writing. The handle is positioned at the beginning of the file. Raises I/O error if the file does not exists.
- **Write Only (‘w’) :** Open the file for writing. For existing file, the data is truncated and over-written. The handle is positioned at the beginning of the file. Creates the file if the file does not exists.
- **Write and Read (‘w+’)** : Open the file for reading and writing. For existing file, data is truncated and over-written. The handle is positioned at the beginning of the file.
- **Append Only (‘a’)** : Open the file for writing. The file is created if it does not exist. The handle is positioned at the end of the file. The data being written will be inserted at the end, after the existing data.
- **Append and Read (‘a+’) :** Open the file for reading and writing. The file is created if it does not exist. The handle is positioned at the end of the file. The data being written will be inserted at the end, after the existing data.

Opening a File

- It is done using the open() function. No module is required to be imported for this function.
- File_object = open(r"File_Name","Access_Mode")
- The file should exist in the same directory as the python program file else, full address of the file should be written on place of filename.

Note: The **r** is placed before filename to prevent the characters in filename string to be treated as special character. For example, if there is \temp in the file address, then \t is treated as the tab character and error is raised of invalid address. The r makes the string raw, that is, it tells that the string is without any special characters. The r can be ignored if the file is in same directory and address is not being placed.

```
# Open function to open the file "MyFile1.txt"
# (same directory) in append mode and
file1 = open("MyFile.txt","a")
# store its reference in the variable file1
# and "MyFile2.txt" in D:\Text in file2
file2 = open(r"D:\Text\MyFile2.txt","w+")
```

Here, file1 is created as object for MyFile1 and file2 as object for MyFile2

Closing a file

close() function closes the file and frees the memory space acquired by that file. It is used at the time when the file is no longer needed or if it is to be opened in a different file mode.

```
# Opening and Closing a file "MyFile.txt"
# for object name file1.
file1 = open("MyFile.txt","a")
file1.close()
```

Writing to a file

There are two ways to write in a file.

- **write() :** Inserts the string str1 in a single line in the text file.File_object.write(str1)
- **writelines() :** For a list of string elements, each string is inserted in the text file.Used to insert multiple strings at a single time.

File_object.writelines(L) for L = [str1, str2, str3]

Reading from a file

There are three ways to read data from a text file.

- **read() :** Returns the read bytes in form of a string. Reads n bytes, if no n specified, reads the entire file.File_object.read([n])
- **readline() :** Reads a line of the file and returns in form of a string.For specified n, reads at most n bytes. However, does not reads more than one line, even if n exceeds the length of the line.File_object.readline([n])
- **readlines() :** Reads all the lines and return them as each line a string element in a list. File_object.readlines()

```
# Program to show various ways to read and  write data in a file.
file1 = open("myfile.txt","w")
L = ["This is Delhi \n","This is Paris \n","This is London \n"]
# \n is placed to indicate EOL (End of Line)
file1.write("Hello \n")
file1.writelines(L)
file1.close() #to change file access modes
file1 = open("myfile.txt","r+")
print "Output of Read function is "
print file1.read()
print
# seek(n) takes the file handle to the nth  bite from the beginning.
file1.seek(0)
print "Output of Readline function is "
print file1.readline()
print
file1.seek(0)
# To show difference between read and readline
print "Output of Read(9) function is "
print  file1.read(9)
print
```

```
file1.seek(0)
print "Output of Readline(9) function is "
print file1.readline(9)
file1.seek(0)
# readlines function
print "Output of Readlines function is "
print file1.readlines()
print
file1.close()
```

output

```
Output of Read function is
Hello
This is Delhi
This is Paris
This is London
```

```
Output of Readline function is
Hello

Output of Read(9) function is
Hello
Th

Output of Readline(9) function is
 Hello

Output of Readline(9) function is
Hello

Output of Readlines function is
['Hello \n', 'This is Delhi \n', 'This is Paris \n', 'This is London \n']
```

Appending to a file

```
# Python program to illustrate  Append vs write mode
file1 = open("myfile.txt","w")
L = ["This is Delhi \n","This is Paris \n","This is London \n"]
file1.close()
# Append-adds at last
file1 = open("myfile.txt","a")#append mode
file1.write("Today \n")
file1.close()
file1 = open("myfile.txt","r")
print "Output of Readlines after appending"
print file1.readlines()
print
file1.close()
# Write-Overwrites
file1 = open("myfile.txt","w")#write mode
file1.write("Tomorrow \n")
file1.close()
file1 = open("myfile.txt","r")
print "Output of Readlines after writing"
print file1.readlines()
print
file1.close()
```

Output of Readlines after appending

```
['This is Delhi \n', 'This is Paris \n', 'This is London \n', 'Today \n']
```

Output of Readlines after writing

```
 ['Tomorrow \n']
```