THE NEW COLLEGE (AUTONOMOUS), CHENNAI-14.

DEPARTMENT OF COMPUTER APPLICATIONS – SHIFT - II

STUDY MATERIAL

SUBJECT: PYTHON PROGRAMMING (20BHM512) **CLASS: III BCA - A**

UNIT-II

2.1 CONTROL FLOW AND SYNTAXN

Programs may contain **set of statements**. These statements are the executable segments that yield the result. In general, statements are executed sequentially, that is the statements are executed one after another. There may be situations in our real life programming where we need to skip a segment or set of statements and execute another segment based on the test of a condition. This is called ***alternative*** or ***branching***.

Also, we may need to execute a set of statements multiple times, called ***iteration*** or ***looping***. In this chapter we are to focus on the various **control structures in Python, their syntax** and learn how to develop the programs using them.

2.2 CONTROL STATEMENTS

A program statement that causes **a jump of control from one part of the program to another** is called **control structure** or **control statement.**

2.2.1 SEQUENTIAL STATEMENT

A **sequential statement** is composed of a **sequence of statements which are executed one after another**. A code to print your name, address and phone number is an example of sequential statement.

Example

```
# Program to print your name and address - example for sequential statement
print ("Hello! This is Kalam")
print ("59, Second Lane, North Car Street, TN")
```

Output

```
Hello! This is Kalam
59, Second Lane, North Car Street, TN
```

2.2.2 ALTERNATIVE OR BRANCHING STATEMENT

In our day-to-day life we need to take various decisions and choose an alternate path to achieve our goal. May be we would have taken an alternate route to reach our destination when we find the usual road by which we travel is blocked. This type of decision making is what we are to learn through alternative or branching statement.

It has 3 types,

i. **Simple if statement**
ii. **if..else statement**
iii. **if..elif statement (Nested if..elif...else)**

(i) Simple if statement

Simple if is the simplest of all decision making statements. Condition should be in the form of relational or logical expression.

Syntax:

if <condition>:

statements-block1

Example

```
# Program to check the age and print whether eligible for voting
x=int (input("Enter your age :"))
if x>=18:
print ("You are eligible for voting")
```

Output

```
Enter your age :34
You are eligible for voting
```

(ii) if..else statement

The **if .. else** statement provides control to check the true block as well as the false block. Following is the syntax of **'if..else'** statement.

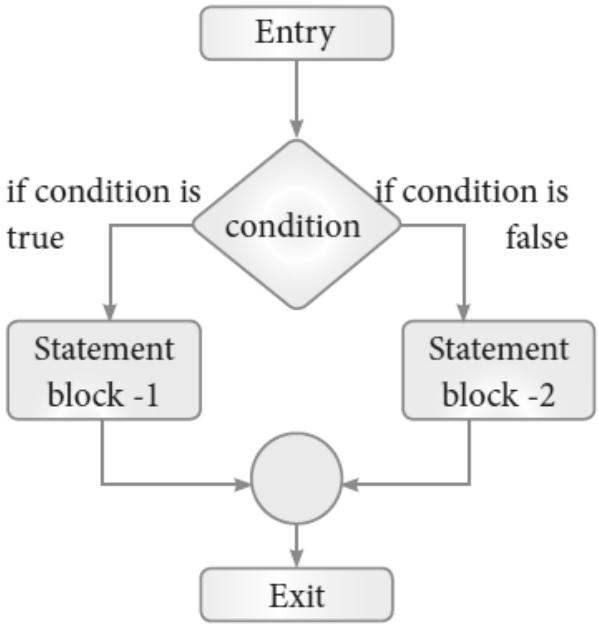
Syntax:

if <condition>:

statements-block 1

else:

statements-block 2



Example:

#Program to check if the accepted number odd or even

```
a = int(input("Enter any number :"))
if a%2= =0:
print (a, " is an even number")
else:
print (a, " is an odd number")
```

Output 1:

```
Enter any number :56
56 is an even number
```

(iii) Nested if..elif...else statement:

When we need to construct a chain of **if** statement(s) then **'elif'** clause can be used instead of **'else'.**

Syntax:

if <condition-1>:

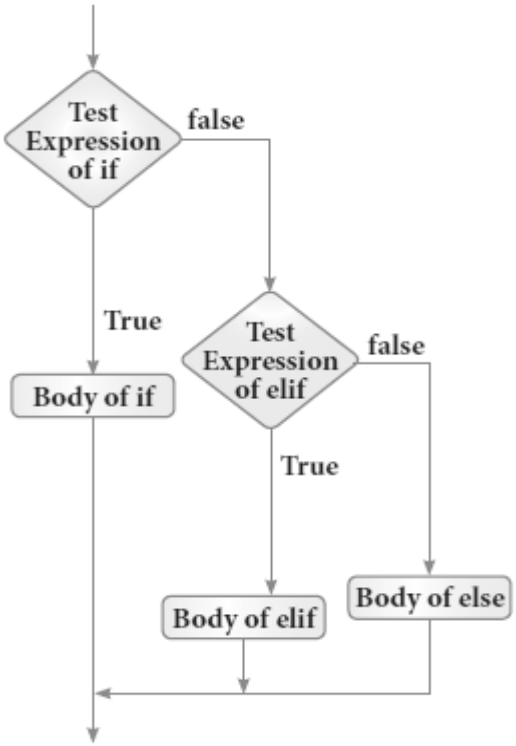
statements-block 1

elif <condition-2>:

statements-block 2

else:

statements-block n



Example:

Average	***Grade***
>=80 and above	*A*
>=70 and <80	*B*
>=60 and <70	*C*
>=50 and <60	D
Otherwise	E

#Program to illustrate the use of nested if statement

```
m1=int (input("Enter mark in first subject : "))
m2=int (input("Enter mark in second subject : "))
avg= (m1+m2)/2
if avg>=80:
print ("Grade : A")
elif avg>=70 and avg<80:
print ("Grade : B")
elif avg>=60 and avg<70:
print ("Grade : C")
elif avg>=50 and avg<60:
print ("Grade : D")
else:
print ("Grade : E")
```

Output 1:

Enter mark in first subject : 34

Enter mark in second subject : 78

Grade : D

2.2.3. ITERATION OR LOOPING CONSTRUCTS

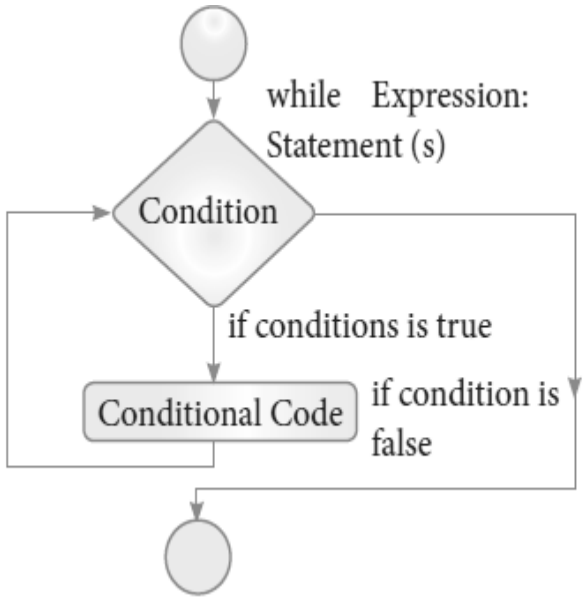
Iteration or loop are used in situation when **the user need to execute a block of code several of times or till the condition is satisfied.** A loop statement allows **executing a statement or group of statements multiple times.**

It has two types of looping constructs:

i) while loop ii) for loop

(i) while loop

In the **while loop**, the condition is any valid Boolean expression returning True or False. The **else** part of while is optional part of **while**. The **statements block1** is kept executed till the condition is True. If the **else** part is written, it is executed when the condition is tested False. Recall **while** loop belongs to entry check loop type, which is it is not executed even once if the condition is tested False in the beginning.



Syntax:

while <condition>:

statements block 1

[else:

statements block2]

Example: program to illustrate the use of while loop - to print all numbers from 10 to 15

```
i=10 # initializing part of the control variable
while (i<=15): # test condition
print (i,end='\t') # statements - block 1
i=i+1 # Updation of the control variable
```

Output:

10 11 12 13 14 15

Example: program to illustrate the use of while loop - with else part

```
i=10                # initializing part of the control variable
while (i<=15):      # test condition
print (i,end='\t')   # statements - block 1
i=i+1               # Updation of the control variable
else:
print ("\nValue of i when the loop exit ",i)
```

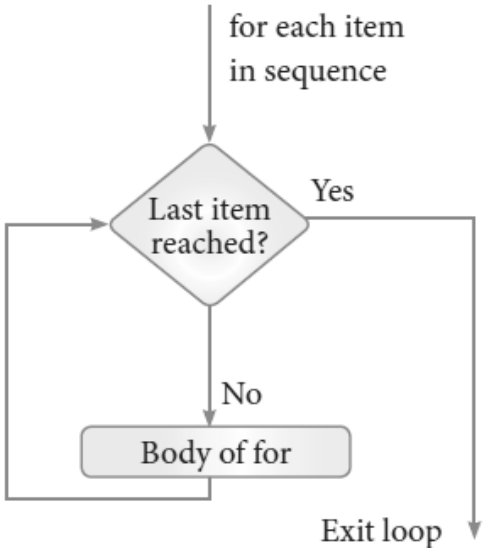
Output: 1

10 11 12 13 14 15

Value of i when the loop exit 16

(ii) for loop

for loop is the **most comfortable loop**. It is also **an entry check** loop. The condition is checked in the beginning and the body of the loop (statements-block 1) is executed if it is only true otherwise the loop is not executed.



Syntax:

for counter_variable in sequence:

statements-block 1

[else: # optional block

statements-block 2]

Usually in python, **for** loop uses the *range()* function in the sequence to specify the **initial, final and increment values**. *range()* generates a list of values starting from **start** till **stop-1.**

The syntax of range() is as follows:

range (start, stop,[step])

Where,

start – refers to the initial value

stop – refers to the final value

step – refers to increment value, this is optional part.

range (1,30,1) *will start the range of values from 1 and end at 29*

range (2,30,2) *will start the range of values from 2 and end at 28*

range (30,3,-3) - *will start the range of values from 30 and end at 6*

range (20) *will consider this value 20 as the end value(or upper limit) and starts the range count from 0 to 19 (remember always range() will work till stop -1value only)*

Example: #program to illustrate the use of for loop - to print single digit even number

```
for i in range (2,10,2):
print (i, end=' ')
```

Output:

2 4 6 8

Example: #program to illustrate the use of for loop - to print single digit even number with else part

```
for i in range(2,10,2):
print (i,end=' ')
else:
print ("\nEnd of the loop")
```

Output:

2 4 6 8

End of the loop

(iii) Nested loop structure

A **loop placed within another loop** is called as nested loop structure. One can place a **while** within another **while; for** within another **for; for** within **while** and **while** within **for** to construct such nested loops.

Example: program to illustrate the use nested loop -for within while loop

```
i=1
while (i<=6):
for j in range (1,i):
print (j,end='\t')
print (end='\n')
i +=1
```

Output:

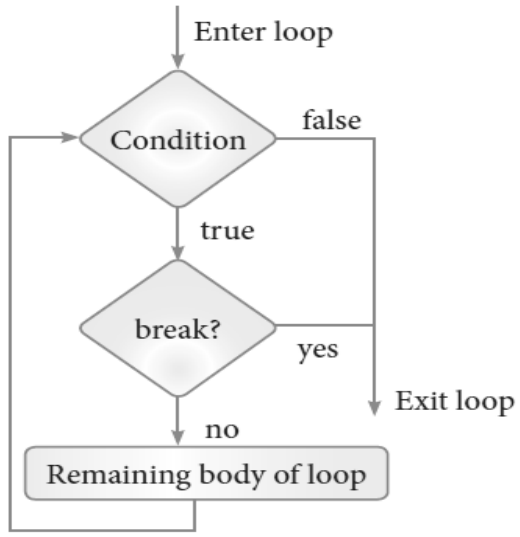
```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

2.2.4 JUMP STATEMENTS IN PYTHON

The jump statement in Python is used to **unconditionally transfer the control from one part of the program to another.** There are three keywords to achieve jump statements in Python: **break, continue, and pass.** The following flowchart illustrates the use of break and continues.

(i) break statement

The **break** statement terminates the loop containing it. Control of the program flows to the statement immediately after the body of the loop. A **while** or **for** loop will iterate till the condition is tested false, but one can even transfer the control out of the loop (terminate) with help of **break** statement.



Syntax:

Break

Example: Program to illustrate the use of break statement inside for loop

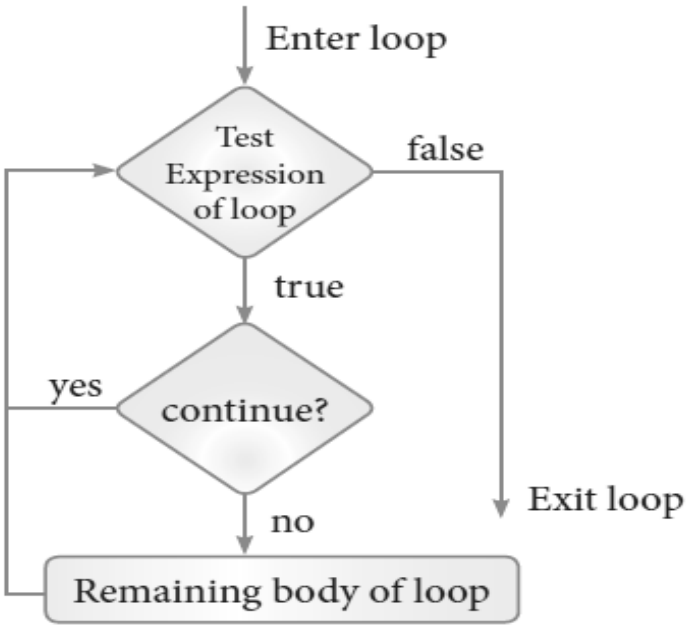
```
for word in “Jump Statement”:
if word = = “e”:
break
print (word, end= ' ')
```

Output:

Jump Stat

(ii) continue statement

Continue statement unlike **the break statement is used to skip the remaining part of a loop** and start with next iteration.



Syntax:

Continue

Example: Program to illustrate the use of continue statement inside for loop

```
for word in “Jump Statement”:
if word = = “e”:
continue
print (word, end = ' ')
print (“\n End of the program”)
```

Output:

Jump Statmnt

End of the program

(iii) pass statement

pass statement in Python programming is **a null statement**. Pass statement when executed by the interpreter it is **completely ignored**. Nothing happens when pass is executed, it results in no operation.

pass statement can be used in **'if'** clause as well as within loop construct, when you do not want any statements or commands within that block to be executed.

Syntax:

pass

Example: Program to illustrate the use of pass statement

```
a=int (input("Enter any number :"))
if (a==0):
pass
else:
print ("non zero value is accepted")
```

Output:

```
Enter any number :3
non zero value is accepted
```

2.3 LIST

(Python programming language has four data types such as **List, Tuples, Set and Dictionary**).

A list in Python is known as a **"sequence data type"** like strings. It is an ordered collection of values enclosed within square brackets [ ]. Each value of a list is called as element. It can be of any type such as numbers, characters, strings and even the nested lists as well. The elements can be modified or mutable which means the elements can be replaced, added or removed. Every element rests at some position in the list. The position of an element is indexed with numbers beginning with zero which is used to locate and access a particular element.

2.3.1 Create a List in Python

In python, a list is simply created by using square bracket. The elements of list should be specified within square brackets.

Syntax:

Variable = [element-1, element-2, element-3 …… element-n]

Example

Marks = [10, 23, 41, 75]

Fruits = [“Apple”, “Orange”, “Mango”, “Banana”]

MyList = [ ]

In the above example, the list Marks has four integer elements; second list Fruits has four string elements; third is an empty list. The elements of a list need not be homogenous type of data. The following list contains multiple type elements.

Mylist = [“Welcome”, 3.14, 10, [2, 4, 6]]

In the above example, Mylist contains another list as an element. This type of list is known as **“Nested List”.** Nested list is a list containing another list as an element.

2.4 TUPLES

Tuples consists of a number of values separated by comma and enclosed within parentheses. Tuple is similar to list, values in a list can be changed but not in a tuple.

2.4.1 CREATING TUPLES

Creating tuples is similar to list. In a list, elements are defined within square brackets, whereas in tuples, they may be enclosed by parenthesis. The elements of a tuple can be even defined without parenthesis. Whether the elements defined within parenthesis or without parenthesis, there is no difference in its function.

Syntax:

 # Empty tuple

Tuple_Name = ( )

 # Tuple with n number elements

Tuple_Name = (E1, E2, E2 ……. En)

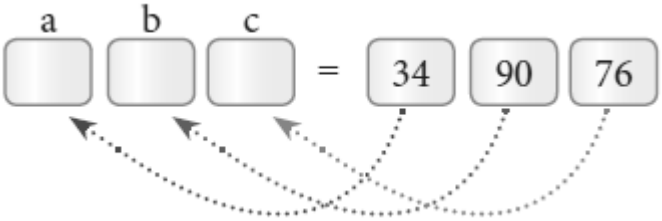
 # Elements of a tuple without parenthesis

Tuple_Name = E1, E2, E3 ….. En

```
>>> MyTup1 = (23, 56, 89, 'A', 'E', 'I', "Tamil")
>>> print(MyTup1)
(23, 56, 89, 'A', 'E', 'I', 'Tamil')
>>> MyTup2 = 23, 56, 89, 'A', 'E', 'I', "Tamil"
>>> print (MyTup2)
(23, 56, 89, 'A', 'E', 'I', 'Tamil')
```

2.4.2 TUPLE ASSIGNMENT

Tuple assignment is a powerful feature in Python. It allows a tuple variable on the left of the assignment operator to be assigned to the values on the right side of the assignment operator. Each value is assigned to its respective variable.



Example

```
>>> (a, b, c) = (34, 90, 76)
>>> print(a,b,c)
34 90 76
# expression are evaluated before assignment
>>> (x, y, z, p) = (2**2, 5/3+4, 15%2, 34>65)
>>> print(x,y,z,p)
4 5.666666666666667 1 False
```

Note that, when you assign values to a tuple, ensure that the number of values on both sides of the assignment operator are same; otherwise, an error is generated by Python.

2.4.3 RETURNING MULTIPLE VALUES IN TUPLES

A function can return only one value at a time, but Python returns more than one value from a function. Python groups multiple values and returns them together.

Example : Program to return the maximum as well as minimum values in a list

```
def Min_Max(n):
a = max(n)
b = min(n)
return(a, b)
Num = (12, 65, 84, 1, 18, 85, 99)
(Max_Num, Min_Num) = Min_Max(Num)
print("Maximum value = ", Max_Num)
print("Minimum value = ", Min_Num)
```

Output:

```
Maximum value = 99
Minimum value = 1
```

2.5 SETS

In python, a set is another type of collection data type. A Set is a mutable and an unordered collection of elements without duplicates. That means the elements within a set cannot be repeated. This feature used to include membership testing and eliminating duplicate elements.

2.5.1 CREATING A SET

A set is created by placing all the elements separated by comma within a pair of curly brackets. The set( ) function can also used to create sets in Python.

Syntax:

Set_Variable = {E1, E2, E3 ……. En}

```
>>> S1={1,2,3,'A',3.14}
>>> print(S1)
{1, 2, 3, 3.14, 'A'}
>>> S2={1,2,2,'A',3.14}
>>> print(S2)
{1, 2, 'A', 3.14}
```

Example

In the above examples, the set S1 is created with different types of elements without duplicate values. Whereas in the set S2 is created with duplicate values, but python accepts only one element among the duplications. Which means python removed the duplicate value, because a set in python cannot have duplicate elements.

2.5.2 CREATING SET USING LIST OR TUPLE

A list or Tuple can be converted as set by using set( ) function. This is very simple procedure. First you have to create a list or Tuple then, substitute its variable within set( ) function as argument.

Example

```
MyList=[2,4,6,8,10]
MySet=set(MyList)
print(MySet)
```

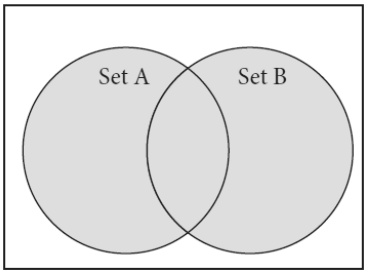
Output:

```
{2, 4, 6, 8, 10}
```

2.5.3 SET OPERATIONS

As you learnt in mathematics, the python is also supports the set operations such as **Union, Intersection**, **difference and Symmetric difference.**

(i) Union: It includes all elements from two or more sets



In python, the operator | is used to union of two sets. The function union( ) is also used to join two sets in python.

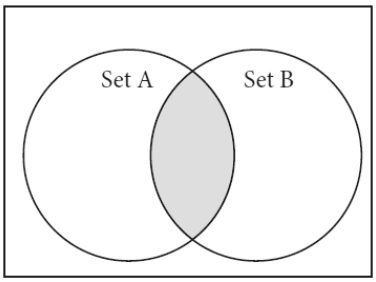
Example: Program to Join (Union) two sets using union operator

```
set_A={2,4,6,8}
set_B={'A', 'B', 'C', 'D'}
U_set=set_A|set_B
print(U_set)
```

Output:

{2, 4, 6, 8, 'A', 'D', 'C', 'B'}

(ii) Intersection: It includes the common elements in two sets



The operator & is used to intersect two sets in python. The function **intersection( )** is also used to intersect two sets in python.

Example: Program to insect two sets using intersection operator

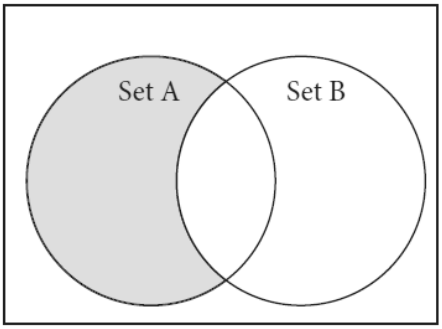
```
set_A={'A', 2, 4, 'D'}
set_B={'A', 'B', 'C', 'D'}
print(set_A & set_B)
```

Output:

{'A', 'D'}

(iii) Difference

It includes all elements that are in first set (say set A) but not in the second set (say set B)



The minus (-) operator is used to difference set operation in python. The function **difference( )** is also used to difference operation.

Example: Program to difference of two sets using minus operator

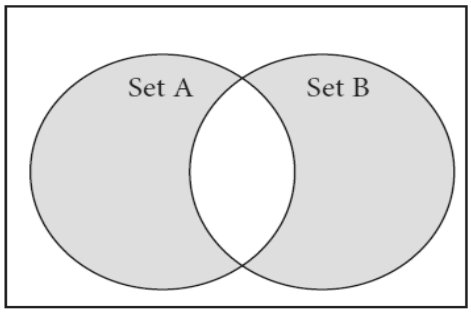
```
set_A={'A', 2, 4, 'D'}
set_B={'A', 'B', 'C', 'D'}
print(set_A - set_B)
```

Output:

{2, 4}

(iv) Symmetric difference

It includes all the elements that are in two sets (say sets A and B) but not the one that are common to two sets.



The caret (^) operator is used to symmetric difference set operation in python. The function **symmetric_difference( )** is also used to do the same operation.

Example: Program to symmetric difference of two sets using caret operator

```
set_A={'A', 2, 4, 'D'}
set_B={'A', 'B', 'C', 'D'}
print(set_A ^ set_B)
```

Output:

{2, 4, 'B', 'C'}

2.6 DICTIONARIES

In python, a dictionary is a mixed collection of elements. Unlike other collection data types such as a list or tuple, the dictionary type stores a key along with its element. The keys in a Python dictionary

is separated by a colon (:) while the commas work as a separator for the elements. The key value pairs are enclosed with curly braces { }.

Syntax of defining a dictionary:

Dictionary_Name = { Key_1: Value_1,

Key_2:Value_2,

…….

Key_n:Value_n

}

Key in the dictionary must be unique case sensitive and can be of any valid Python type.

2.6.1 Creating a Dictionary

 # Empty dictionary

Dict1 = { }

 # Dictionary with Key

Dict_Stud = { 'RollNo': '1234', 'Name':'Murali', 'Class':'XII', 'Marks':'451'}

2.6.2 Dictionary Comprehensions

In Python, comprehension is another way of creating dictionary. The following is the syntax of creating such dictionary.

Syntax

Dict = { expression for variable in sequence [if condition] }

The if condition is optional and if specified, only those values in the sequence are evaluated using the expression which satisfy the condition.

Example

*Dict = { x : 2 * x for x in range(1,10)}*

Output

{1: 2, 2: 4, 3: 6, 4: 8, 5: 10, 6: 12, 7: 14, 8: 16, 9: 18}

2.6.3 Accessing, Adding, Modifying and Deleting elements from a Dictionary

Accessing all elements from a dictionary is very similar as Lists and Tuples. Simple print function is used to access all the elements. If you want to access a particular element, square brackets can be used along with key.

Example : Program to access all the values stored in a dictionary

```
MyDict = { 'Reg_No': '1221',
'Name' : 'Tamilselvi',
'School' : 'CGHSS',
```

```
'Address' : 'Rotler St., Chennai 112' }
print(MyDict)
print("Register Number: ", MyDict['Reg_No'])
print("Name of the Student: ", MyDict['Name'])
print("School: ", MyDict['School'])
print("Address: ", MyDict['Address'])
```

Output:

```
{'Reg_No': '1221', 'Name': 'Tamilselvi', 'School': 'CGHSS', 'Address': 'Rotler St., Chennai 112'}
Register Number: 1221
Name of the Student: Tamilselvi
School: CGHSS
Address: Rotler St., Chennai 112
```

Example : Program to add a new value in the dictionary

```
MyDict = { 'Reg_No': '1221',
'Name' : 'Tamilselvi',
'School' : 'CGHSS', 'Address' : '
Rotler St., Chennai 112'}
print(MyDict)
print("Register Number: ", MyDict['Reg_No'])
print("Name of the Student: ", MyDict['Name'])
MyDict['Class'] = 'XII - A' # Adding new value
print("Class: ", MyDict['Class']) # Printing newly added value
print("School: ", MyDict['School'])
print("Address: ", MyDict['Address'])
```

Example : Program to delete elements from a dictionary and finally deletes the dictionary.

```
Dict = {'Roll No' : 12001, 'SName' : 'Meena', 'Mark1' : 98, 'Marl2' : 86}
print("Dictionary elements before deletion: \n", Dict)
del Dict['Mark1'] # Deleting a particular element
print("Dictionary elements after deletion of a element: \n", Dict)
Dict.clear() # Deleting all elements
print("Dictionary after deletion of all elements: \n", Dict)
del Dict
print(Dict) # Deleting entire dictionary
```

Output:

Dictionary elements before deletion:

{'Roll No': 12001, 'SName': 'Meena', 'Mark1': 98, 'Marl2': 86}

Dictionary elements after deletion of a element:

{'Roll No': 12001, 'SName': 'Meena', 'Marl2': 86}

Dictionary after deletion of all elements:

{ }

Traceback (most recent call last):

File "E:/Python/Dict_Test_02.py", line 8, in <module>

print(Dict)

NameError: name 'Dict' is not defined

~~~~~~ The End of Unit - II ~~~~~~