Dr. D. Y. PATIL INSTITUTE OF TECHNOLOGY, PIMPRI, PUNE-18

Department of First Year Engineering

Programming and Problem Solving

Unit- 2 Notes

Unit II: Decision Control Statements

2.1 Selection/conditional Statements:

Q. Explain selection/conditional statements with flowchart

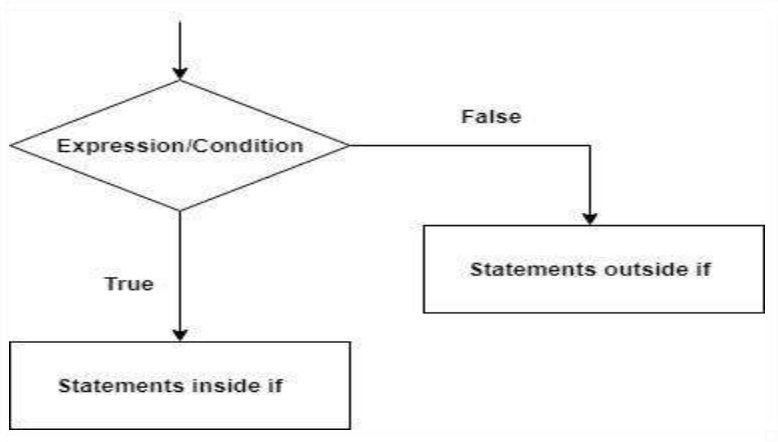
- The selection control statements usually jump from one part of code to another depending on whether a particular condition is satisfied or not.
- It allow us to execute statements selectively (branch wise) based on certain decisions.
- Python language supports different types of selection/conditional/branching statements.
    1. If statements
    2. If....else statements
    3. If…elif….else statements
    4. Nested if

2.1.1 if

1. **If statements:**
    - If statement is a simplest form of decision control statements that is frequently used in decision making.
    - If statements may include one statement or N statements enclosed within the if block
    - If the expression is true the statements of if block get executed, otherwise these statements will be skipped and the execution will be jump to statement just outside if block.
    - **Syntax:**

```
if (expression/condition):
    Block of statements
Statements ouside if
```

- **Flowchart:**



✓ **Example:**

```
a = 2
if (a<3):
    print(a);
if (a>3):
    print('Hi')
```

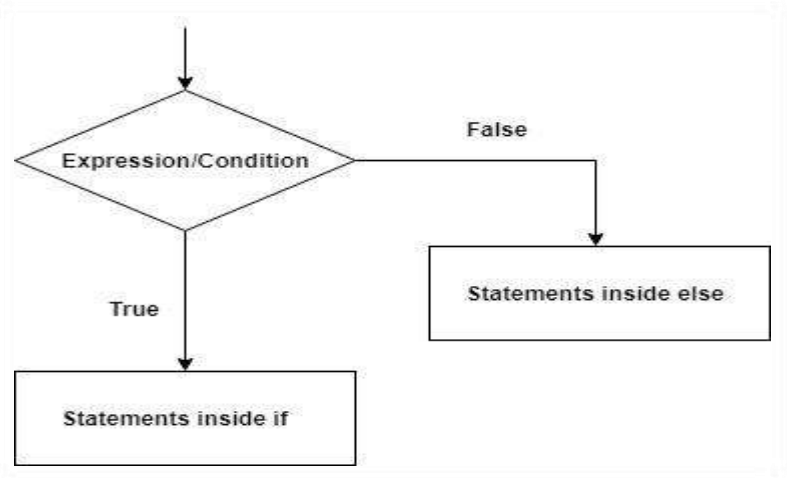
Output: **2**

2.1.2 if-else

- In simple if statement, test expression is evaluated and if it is true the statements in if block get executed, But if we want a separate statements to be get executed if it returns false in this case we have to use if..else control statement.
- if else statement has two blocks of codes, each for if and else.
- The block of code inside if statement is executed if the expression in if statement is TRUE, else the block of code inside else is executed.
- **Syntax:**

```
if (expression/condition):
        Block of statements
else:
        Block of statements
```

- **Flowchart:**



✓ **Example:**

```
x = 5
if (x<10):
    print ("Condition is TRUE")
else:
    print ("Condition is FALSE")

output:
        Condition is TRUE'
```

2.1.3 if-elif-else

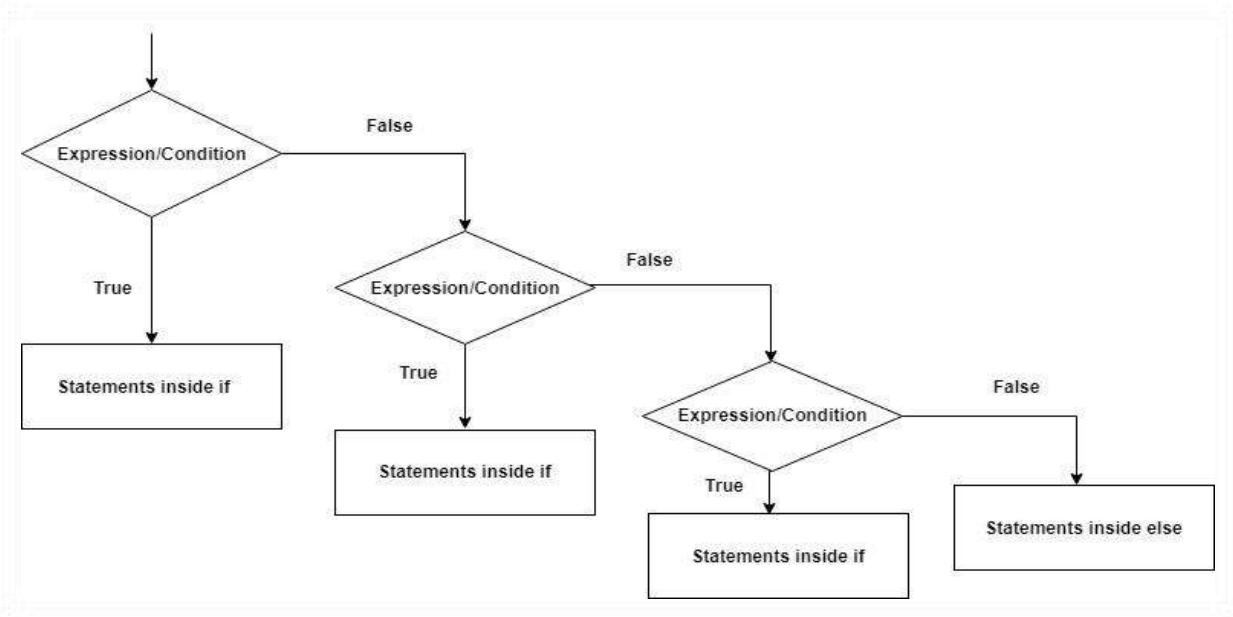
- Python supports if..elif..else to test additional conditions apart from the initial test expression.
- It work just like a if…else statement.
- The programmer can have as many elif statements/branches as he wants depending on the expressions that have to be tested.
- A series of if..elif statements can have a final else block, which is called if none of the if or elif expression is true.

- **Syntax:**

```
if (expression/condition):
    Block of statements
```

elif (expression/condition):
    Block of statements
elif (expression/condition):
    Block of statements
else:
    Block of statements

- **Flowchart:**



Example:

```
x = 5
if (x == 2):
  print ("x is equal to 2")
elif (x < 2):
  print ("x is less than 2")
else:
  print ("x is greater than 2")
```

Output:

x is greater than 2

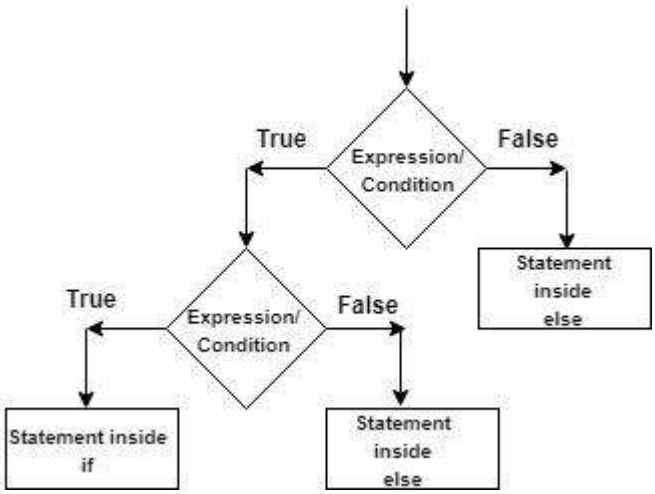
2.1.4 nested if

- A statement that contains other statements is called nested/compound statements.
- In nested if else statements, if statement is nested inside an if statements. So, the nested if statements will be executed only if the expression of main if statement returns TRUE.

- **Syntax:**

```
if (expression/condition):
        if (expression/condition):
            statements
        else:
            statements
else:
    statements
```

- **Flowchart**



- **Example**

```
x = 5
if (x<=10):
    if(x==10):
        print ("Number is Ten")
    else:
        print ("Number is less than Ten")
else:
    print ("Number is greater than Ten")
```

2.2 Basic loop Structures/Iterative statements:

Q. Explain looping statements with flowchart.

- Iterative statements are decision control statements that are used to repeat the execution of a list of statements.
- Python language supports two types of statements while loop and for loop.

2.2.1 while loop

- The while loop provides a mechanism to repeat one or more statements while a particular condition is True.
- **Syntax:**

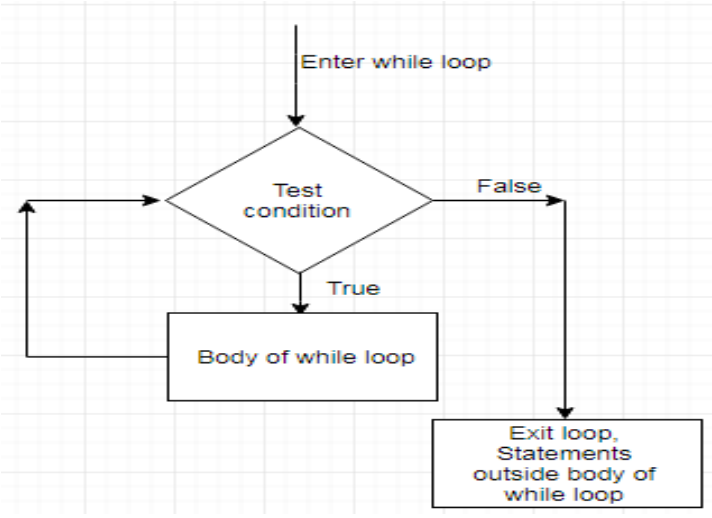
while condition:

Statement block

Statement y

- As shown in the syntax above, in the while loop condition is first tested.
- If the condition is True, only then the statement block will be executed.
- Otherwise, if the condition is False the control will jump to statement y, that is the immediate statement outside the while loop block.

- **Flowchart**:



- **Example:**
  Program to print numbers from 0 to 10.

```
i=0
while (i<10):
        print(i, end=" ")
        i=i+1
```

Output: 0 1 2 3 4 5 6 7 8 9

2.2.2 for loop

- The for loop provide a mechanism to repeat a task until a particular condition is True.
- The for loop is usually known as a determinate or definite loop because the programmer knows exactly how many times the loop will repeat.
- The for loop in python is used to iterate through each item in a sequence.
- Here, by sequence we mean just an ordered collection of items.
- The sequence of for loop is tested; if it is satisfied then body of for loop is executed.
- When the last item in sequence is reached, then for loop exits.

- **Syntax:**

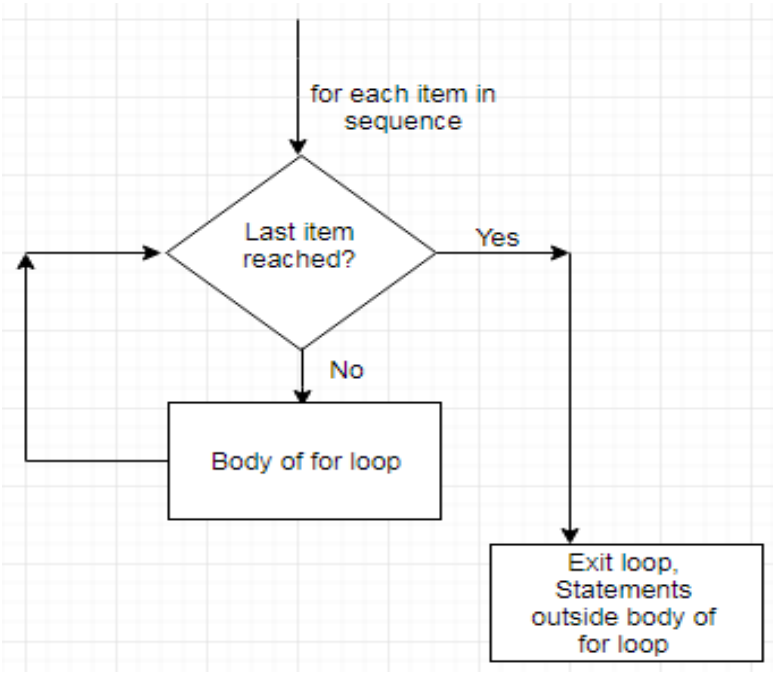
```
for loop_control_var  in sequence:
        statement block1
statement x
```

- **The range() function: (Optional)**
  - The range() function is inbuilt function in python that is used to iterate over a sequence of numbers.
  - The syntax of range() is

range(beginning, end, [step])

The range( ) function produces a sequence of numbers starting with beginning(inclusive) and ending with one less than the number end. The step argument is optional

- **Flowchart**:



- **Example:**.

for i in range(1, 5): print(i, end=" ") Output: 1 2 3 4	for i in range(1, 10, 2): print(i, end=" ") Output: 1 3 5 7 9

2.2.3 Selecting appropriate loop:

- Loops can be of different types such as entry controlled (also known as pre-test), exit controlled (also known as post-test), counter controlled and condition controlled (or sentinel-controlled) loops.

Pre-test and Post-test loops:

- While in an entry controlled loop, condition is tested before the loop starts, an exit-controlled loop, tests the condition after the loop is executed.

- If condition is not met in entry-controlled loop, then the loop will never execute.
- However, in case of post-test, the body of the loop is executed unconditionally for the first time.
- If the requirement is to have a pre-test loop, then choose a for loop or a while loop.
- The table below shows comparison between ore-test loop and post-test loop.

Feature	Pre-test Loop	Post-test Loop
Initialization	1	2
Number of tests	N+1	N
Statements executed	N	N
Loop control variable update	N	N
Minimum Iterations	0	1

Condition-controlled and Counter-controlled loops:

- When we know in advance the number of times the loop should be executed, we use a counter controlled loop.
- The counter is a variable that must be initialized, tested and updated for performing the loop operations.
- Such a counter controlled loop in which the counter is assigned a constant or a value is known as a definite repetition loop.
- When we do not know in advance the number of times the loop will be executed, we use a condition-controlled (or sentinel-controlled or indefinite loop) loop.
- In such a loop, a special value called a sentinel value is used to change the loop control expression from true to false.
- For example, when data is read from user, the user may be notified that when they want the execution to stop, they may enter -1. This value is called sentinel value.
- If your requirement is to have a counter-controlled loop, then choose for loop, else, if you need to have a sentinel controlled loop then go for a while loop.

Comparison between condition-controlled and counter controlled loop:

Attitude	Counter-controlled loop	Condition Controlled loop
Number of execution	Used when number of times the loop has to be executed is known in advance.	Used when number of times the loops has to be executed is not known in advance.
Condition Variable	In counter-controlled loops, we have a counter variable.	In condition-controlled loops, we use a sentinel variable.
Value and limitation of variable	The value of the counter variable and the condition for loop execution, both are strict.	The value of the counter variable and the condition for loop execution, both are strict.

Example	i=0
	while (i<=10):
	print(i,end= " ")
	i+=1
		i=1
		while(i>0):
		print(i,end= " ")
		i+=1
		if (i==10):
		break

2.3 Nested loops

- Python allows its users to have a nested loop, that is, loops can be placed inside another loop.
- This feature works with while as well as for loop.
- Nested loops are commonly used with for loop because it is easiest to control.
- Loops can be nested at any desired level.
- Loops should be properly indented to identify which statements are contained within which loop.
- **Example:**

```
for i in range(0, 5):
    for j in range(0, i+1):
```

```
        print("* ",end="")
    print("\r")
```

Output

```
*
* *
* * *
* * * *
* * * * *
```

Example 2:

```
lastNumber = 6
for row in range(1, lastNumber):
    for column in range(1, row + 1):
        print(column, end=' ')
    print("")
```

Output

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

2.4 else statement used with loops

- In for loop, the else keyword is used to specify a block of code to be executed after completion of loop.
- If the else statement is used with a for loop, the else statement is executed when the loop has completed iterating.
- But when used with the while loop, the else statement is executed when the condition becomes false.

- Example 1 (For 'for loop'): Print numbers from 0 to 10 and print a message when the loop is ended.

```
for x in range(11):
print(x)
else:
print("Loop is finished")
```

- Example 2 (For 'while loop"):

```
i=1
while(i<0)
print(i)
i=i-1
else:
print(i, "is not negative so loop did not execute")
```

Output:

1 is not negative so loop did not execute.

2.5 Branching Statements:

Q. Write a note on Break, pass and continue.

2.5.1 break

- The break statement is used to terminate the execution of the nearest enclosing loop in which it appears.
- The break is widely used with *for* and *while* loop.
- When the break statement is encountered inside a loop, the loop is immediately terminated and program control is passed to next statement following the loop.
- The syntax of break is as shown below:
- **Syntax:**

```
break
```

- The break statement is used to exit a loop from any point within its body, bypassing its normal termination expression.
- Example:

```
for i in "welcome":
```

```
if(i=="c"):
    break
print(i)
```

Output:

```
w
e
l
```

2.5.2 continue

- Continue statement can appear only in the body of loop.
- When the continue statement encounters, then the rest of the statements in the loop are skipped
- The control is transferred to the loop-continuation portion of the nearest enclosing loop.
- **Its syntax is simple just type keyword continue as shown below**

```
continue
```

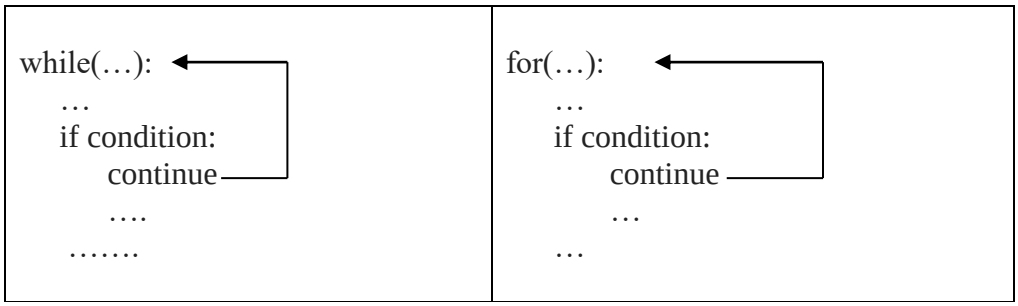
- **Example:**

```
for i in "welcome":
    if(i=="c"):
        continue
    print(i)
```

Output:

```
w
e
l
o
m
e
```

- We can say continue statement forces the next iteration of the loop to take place, skipping the remaining code below continue statement.



❖ Differentiate Between break & continue:

	Break	Continue
Task	It terminates the execution of remaining iteration of the loop.	It terminates only the current iteration of the loop.
Control after statement	When 'break' statement appears the control goes out of the loop.	When 'continue' statement appears the control goes at the beginning of the loop.
Causes	It causes early termination of loop.	It causes early execution of the next iteration.
Continuation	Break stops the continuation of loop.	Continue do not stop the continuation of loop.
Syntax	break	continue

Example	for i in 'welcome':
	if (i=='c')
	break
	print(i)
	Output:
	w
	e
	l
		for i in 'welcome':
		if (i=='c')
		continue
		print(i)
		Output:
		w
		e
		l
		o
		m
		e

2.5.3 pass

- It is used when a statement is required syntactically but you do not want any command or code to execute.
- The pass statement is a null operation.
- Nothing happens when it executes.

- It is used as placeholder.
- In a program where for some conditions you don't want to execute any action or what action to be taken is not yet decided, but we may wish to write some code in future, In such cases pass can be used.
- **Syntax:**

  Pass

- **Example:**

```
for i in "welcome":
  if (i == 'c'):
      pass
  print (i)
```

Output:

```
w
e
l
c
o
m
e
```

2.6 Other data types

2.6.1 Tuples

Q. What is tuple? Explain how to access, add and remove elements in a tuple

- Tuple is a data structure supported by python.
- A tuple is a sequence of immutable objects. That means you cannot change the values in a tuple.
- Tuples use parentheses to define its elements.

Accessing elements in a Tuple:

- Indices in a tuple start at 0.
- We can perform operations like slice, concatenate etc. on a tuple.

- For example, to access values in tuple, slice operation is used along with the index or indices to obtain value stored at that index.
- Example:

```
Tup1=(1,2,3,4,5,6,7,8,9,10)

print("Tup[3:6]=",Tup1[3:6])

print("Tup[:8]=",Tup1[:4])

print("Tup[4:]=", Tup1[4:])
```

output:

Tup[3:6]=(4,5,6)

Tup[:8]=(1,2,3,4)

Tup[4:]=(5,6,7,8,9,10)

Adding Elements in a Tuple:

- Tuples are immutable means we cannot add new element into it.
- Also it is not possible to change the value of tuple.

Program:

```
tup1=("Kunal","Ishita","Shree","Pari","Rahul")
tup1[5]="Raj"
print(tup1)
```

Output:

It will raise an error because new element cannot be added.

Removing/Deleting elements in Tuple:

- Since tuple is an immutable data structure, you cannot delete value(s) from it.

Program:

Tup1=(1,2,3,4,5)

del Tup1[3]

print(Tup1)

output: It will raise error as 'tuple' object doesn't support item deletion.

- However, we can delete the entire tuple by using the "del" statement.

Example:

Tup1=(1,2,3,4,5)

del Tup1

Operations on Tuple

Q. Explain various operations on Tuple.

- Tuple behave like similar way as string when operators like concatenation (+), and repetition (*) are used and returns new list rather a string.
- Tuple supports various operations as listed below:

Operation	Description	Example	Output
Concatenation(+)	-It joins two list and returns new list.	X=(10,11,12) Y=(13,14, "pune") Z=X+Y print(z)	(10,11,12,13,14, "pune")
Repetition(*)	-It repeats elements from the list.	X=(10,11,12) Z=X*2 print(z)	(10,11,12,10,11,12)
Slice []	- It will give you element from a specified index.	Example: List2 = ('John','PPS',94) print(List2[1])	PPS
Range slice[:]	-It will give you elements from specified range slice. -Range slice contains inclusive and exclusive slices	X=(10,11,12,13,14, 15) Z=X[0:3] print(z)	(10,11,12)

Memebership(in)	-It will check whether given value present in list or not. – accordingly it will return Boolean value	X=(10,11,12,13,14, 15) print(11 in X)	TRUE
Length – len()	-It will return length of given list	X=(10,11,12,13,14, 15) print(len(X))	6
Maximum – max()	-It will return maximum value from list	X=(10,11,12,13,14, 15) print(max(X))	15
Minimum()-min()	-It will return minimu value from list	X=(10,11,12,13,14, 15) print(min(X))	10

2.6.2 Lists

Q. What is list? Explain how elements are accessed, added and removed from list.

- List is a data type in python.
- List is a collection of values (items / elements).
- The items in the list can be of different data types.
- The elements of the list are separated by comma (,) and written between square brackets [].
- List is mutable which means the value of its elements can be changed.
- Syntax:

  List = [value1, value2, …]

- Example1:

  List1 = [1,2,3,4]

  print(List1)

  Output:

  [1, 2, 3, 4]

- Example 2:

  List2 = ['John','PPS',94]

  print(List2)

Output:

['John','PPS',94]

➢ Accessing Values in List:

- Elements in the list can be accessed by their index.
- Starting index of list is 0 (zero).
- To access values in list, square brackets [] are used to slice and index is written inside it.
- Example:

```
List2 = ['John','PPS',94]
print(List2[1])
print(List2[0:2])
```

Output:

```
PPS
['John', 'PPS']
```

➢ Adding elements / Updating Values in List

- One or more elements can be updated in the list by giving slice on the left hand side of assignment operator.
- New values can be appended in the list using append().
- New element is given as parameter to append function.
- Example:

```
List2 = ['John','PPS',94]
List2.append('FE')
print(List2)
List2[0]="Sam"
print(List2)
```

Output:

```
['John', 'PPS', 94, 'FE']
['Sam', 'PPS', 94, 'FE']
```

➢ Removing / Deleting elements from List

- Element can be removed from list using del statement or remove().

- Here, index of element or name of element is provided as a argument to remov().
- Example:

```
List2 = ['John', 'PPS', 94, 'FE']
print(List2)
del List2[1]
print(List2)
List2.remove(List2[0])
print(List2)
List2.remove(94)
print(List2)
```

Output:

```
['Sam', 'PPS', 94, 'FE']
['Sam', 94, 'FE']
[94, 'FE']
['FE']
```

Operations on list

Q. Explain various operations on list.

- List behave like similar way as string when operators like concatenation (+), and repetition (*) are used and returns new list rather a string.
- List supports various operations as listed below:

Operation	Description	Example	Output
Concatenation(+)	-It joins two list and returns new list.	X=[10,11,12] Y=[13,14, “pune”] Z=X+Y print(z)	[10,11,12,13,14, “pune”]
Repetition(*)	-It repeats elements from the list.	X=[10,11,12] Z=X*2 print(z)	[10,11,12,10,11,12]
Slice []	- It will give you element from a specified index.	Example: List2 = ['John','PPS',94] print(List2[1])	PPS

Range slice[:]	-It will give you elements from specified range slice. -Range slice contains inclusive and exclusive slices	X=[10,11,12,13,14, 15] Z=X[0:3] print(z)	[10,11,12]
Memebership(in)	-It will check whether given value present in list or not. – accordingly it will return Boolean value	X=[10,11,12,13,14, 15] print(11 in X)	TRUE
Length – len()	-It will return length of given list	X=[10,11,12,13,14, 15] print(len(X))	6
Maximum – max()	-It will return maximum value from list	X=[10,11,12,13,14, 15] print(max(X))	15
Minimum()-min()	-It will return minimu value from list	X=[10,11,12,13,14, 15] print(min(X))	10

2.6.3 Dictionary

Q. What is dictionary? Explain how to create dictionary, access, add and remove elements from dictionary.

- Dictionary is a data structure which stores elements as key: value pairs.
- Each key is separated from its value by a colon (:).
- Consecutive key: value pairs are separated by commas (,).
- The entire items in a dictionary are enclosed in curly brackets {}.
- Syntax:

  Dict = {key1: value1, key2: value2, key3: value3}
- If there are many key: value pairs in dictionary, then we can write just one key: value pair on one line to make code easier to read and understand.

  Dict = {key1: value1, key2: value2, key3: value3……. keyN: valueN.}

- Keys can be of any data type.
- The keys in dictionary must be unique.
- Dictionary keys are case sensitive.

➢ **Creating Dictionary**

- The syntax to create an empty dictionary can be given as:

  Dict = { }
- The syntax to create a dictionary with key value pair is:

  Dict = {key1: value1, key2: value2, key3: value3}
- Example1:

  Dict = { }

  Print(Dict)

  Output:

  {}
- Example2:

  Dict = {1: "PPS", 2: "PHY"}

  print(Dict)

  Output:

  {1: 'PPS', 2: 'PHY'}

➢ **Accessing values from Dictionary**

- To access values in dictionary, square brackets are used along with the key to obtain its value.
- Example:

  Dict = {1: "PPS", 2: "PHY"}

  print(Dict[1])

  Output:

  PPS
- Note that if you try to access an item with a key, which is not specified in dictionary, a KeyError is generated.

- **Adding and Modifying (Updating) an item in Dictionary**
  - Following syntax is used to add a new key: value pair in a dictionary.
  - Syntax

    Dict[key] = value
  - Example

    Dict = {1: "PPS", 2: "PHY"}

    Dict[3] = "M-I"

    print(Dict)

    Output

    {1: 'PPS', 2: 'PHY', 3: 'M-I'}
  - To modify an entry, just overwrite existing value as shown below:
  - Example

    Dict = {1: "PPS", 2: "PHY"}

    Dict[2] = "SME"

    print(Dict)

    Output

    {1: 'PPS', 2: 'SME'}

- **Deleting items from Dictionary**
  - One or more items of dictionary can be deleted using del keyword.
  - To remove all items in dictionary in just one line, clear() is used.
  - To remove entire dictionary from memory, del statement can be used as del Dict.
  - Example:

    Dict = {1: "PPS", 2: "PHY", 3: "SME"}

    del Dict[2]

    print(Dict)

    Dict.clear()

    print(Dict)

    del Dict

    Output

```
{1: 'PPS', 3: 'SME'}
{}
```

Operations on Dictionary:

Method	Description
clear()	Remove all items form the dictionary.
copy()	Return a shallow copy of the dictionary.
items()	Return a new view of the dictionary's items (key, value).
keys()	Return a new view of the dictionary's keys.
values()	Return a new view of the dictionary's values