Unit 2

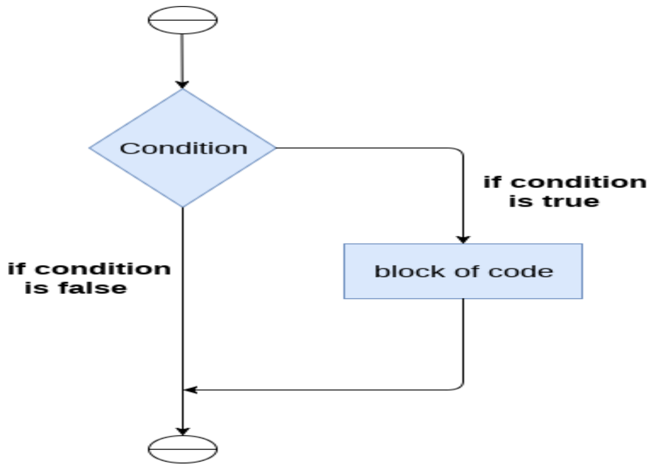
Decision Making Statements

If-else statements

- Decision making allows us to run a particular block of code for a particular decision.
- Here, the decisions are made on the validity of the particular conditions. Condition checking is the backbone of decision making.
- If statement is used to test a specific condition. If the condition is true, a block of code (if-block) will be executed.
- if-else statement is similar to if statement except the fact that, it also provides the block of the code for the false case of the condition to be checked. If the condition provided in the if statement is false, then the else statement will be executed.
- Nested if statements enable us to use if ? else statement inside an outer if statement.

If statement

The if statement is used to test a particular condition and if the condition is true, it executes a block of code known as if-block. The condition of if statement can be any valid logical expression which can be either evaluated to true or false.



The syntax of the if-statement is given below.

```
1. if expression:
2.     statement
```

Example 1

```
1. num = int(input("enter the number?"))
2. if num%2 == 0:
3.     print("Number is even")
```

Output:

```
enter the number?10
Number is even
```

Example 2 : Program to print the largest of the three numbers.

```
a = int(input("Enter a? "));
b = int(input("Enter b? "));
c = int(input("Enter c? "));
if a>b and a>c:
    print("a is largest");
if b>a and b>c:
    print("b is largest");
if c>a and c>b:
    print("c is largest");
```

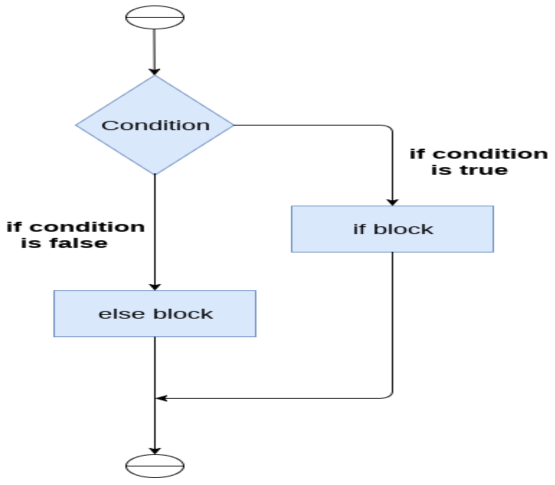
Output:

```
Enter a? 100
Enter b? 120
Enter c? 130
c is largest
```

If-else statement

The if-else statement provides an else block combined with the if statement which is executed in the false case of the condition.

If the condition is true, then the if-block is executed. Otherwise, the else-block is executed.



The syntax of the if-else statement is given below.

```
if condition:
    #block of statements
else:
    #another block of statements (else-block)
```

Example 1 : Program to check whether a person is eligible to vote or not.

```
age = int (input("Enter your age? "))
if age>=18:
    print("You are eligible to vote !!");
else:
    print("Sorry! you have to wait !!");
```

Output:

```
Enter your age? 90
You are eligible to vote !!
```

Example 2: Program to check whether a number is even or not.

```
num = int(input("enter the number?"))
if num%2 == 0:
    print("Number is even...")
else:
    print("Number is odd...")
```

Output:

```
enter the number?10
Number is even
```

elif statement:-

The elif statement enables us to check multiple conditions and execute the specific block of statements depending upon the true condition among them. We can have any number of elif statements in our program depending upon our need. However, using elif is optional.

The elif statement works like an if-else-if ladder statement in C. It must be succeeded by an if statement.

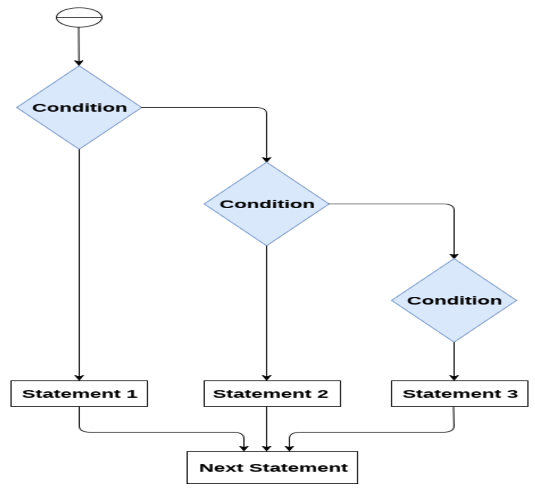
The syntax of the elif statement is given below.

```
if expression 1:
    # block of statements

elif expression 2:
    # block of statements

elif expression 3:
    # block of statements

else:
    # block of statements
```



Example 1

```
number = int(input("Enter the number?"))
if number==10:
    print("number is equals to 10")
elif number==50:
    print("number is equal to 50");
elif number==100:
    print("number is equal to 100");
else:
    print("number is not equal to 10, 50 or 100");
```

Output:

```
Enter the number?15
number is not equal to 10, 50 or 100
```

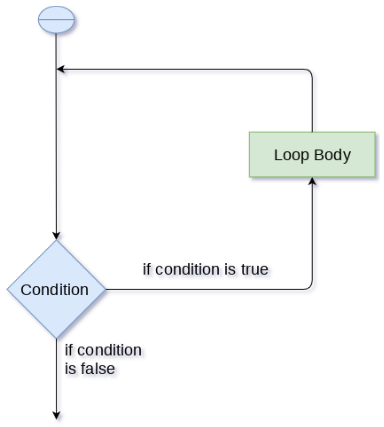
Example 2

```
marks = int(input("Enter the marks? "))
f marks > 85 and marks <= 100:
   print("Congrats ! you scored grade A ...")
lif marks > 60 and marks <= 85:
   print("You scored grade B + ...")
lif marks > 40 and marks <= 60:
   print("You scored grade B ...")
lif (marks > 30 and marks <= 40):
   print("You scored grade C ...")
lse:
   print("Sorry you are fail ?")
```

Loops:-

The flow of the programs written in any programming language is sequential by default. Sometimes we may need to alter the flow of the program. The execution of a specific code may need to be repeated several numbers of times.

For this purpose, The programming languages provide various types of loops which are capable of repeating some specific code several numbers of times. Consider the following diagram to understand the working of a loop statement.



The looping simplifies the complex problems into the easy ones. It enables us to alter the flow of the program so that instead of writing the same code again and again, we can repeat the same code for a finite number of times. For example, if we need to print the first 10 natural numbers then, instead of using the print statement 10 times, we can print inside a loop which runs up to 10 iterations.

Advantages of loops

There are the following advantages of loops in Python.

1. It provides code re-usability.
2. Using loops, we do not need to write the same code again and again.
3. Using loops, we can traverse over the elements of data structures (array or linked lists).

There are the following loop statements in Python.

- ❖ **for loop** - The for loop is used in the case where we need to execute some part of the code until the given condition is satisfied. The for loop is also called as a per-tested loop. It is better to use for loop if the number of iteration is known in advance.
- ❖ **while loop** - The while loop is to be used in the scenario where we don't know the number of iterations in advance. The block of statements is executed in the while loop until the condition specified in the while loop is satisfied. It is also called a pre-tested loop.
- ❖ **do-while loop** - The do-while loop continues until a given condition satisfies. It is also called post tested loop. It is used when it is necessary to execute the loop at least once (mostly menu driven programs).

for loop:-

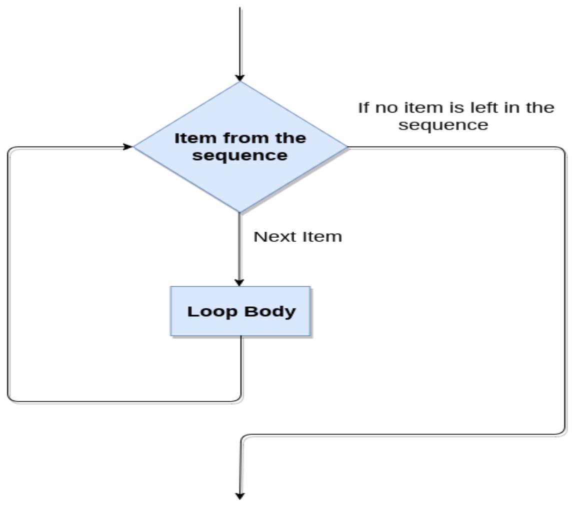
for loop in Python is used to iterate the statements or a part of the program several times. It is frequently used to traverse the data structures like list, tuple, or dictionary.

The syntax of for loop in python is given below.

statement(s)

for iterating_var **in** sequence: Example

```
i=1
n=int(input("Enter the number up to which you want to print the natural numbers?"))
for i in range(0,10):
    print(i,end = ' ')
```



Output:

```
0 1 2 3 4 5 6 7 8 9
```

Example 2 : printing the table of the given number

```
i=1;
num = int(input("Enter a number:"));
for i in range(1,11):
    print("%d X %d = %d"%(num,i,num*i));
```

Output:

```
Enter a number:10
10 X 1 = 10
10 X 2 = 20
10 X 3 = 30
10 X 4 = 40
10 X 5 = 50
10 X 6 = 60
10 X 7 = 70
10 X 8 = 80
10 X 9 = 90
10 X 10 = 100
```

Nested for loop:-

Python allows us to nest any number of for loops inside a for loop. The inner loop is executed n number of times for every iteration of the outer loop.

Syntax of the nested for loop

```
for iterating_var1 in sequence:
        for iterating_var2 in sequence:
                #block of statements
          #Other statements
```

Example 1

```
n = int(input("Enter the number of rows you want to print?"))
i,j=0,0
for i in range(0,n):
    print()
    for j in range(0,i+1):
        print("*",end="")
```

Output:

```
Enter the number of rows you want to print?5
*
**
***
****
*****
```

for loop with else statement

Unlike other languages like C, C++, or Java, python allows us to use the else statement with the for loop which can be executed only when all the iterations are exhausted. Here, we must notice that if the loop contains any of the break statement then the else statement will not be executed.

Example 1

```
for i in range(0,5):
    print(i)
else:print("for loop completely exhausted, since there is no break.");
```

In the above example, for loop is executed completely since there is no break statement in the loop. The control comes out of the loop and hence the else block is executed.

Output:

```
0
1
2
3
4
for loop completely exhausted, since there is no break.
```

Example 2

```
for i in range(0,5):
   print(i)
   break;
else:print("for loop is exhausted");
print("The loop is broken due to break statement...came out of loop")
```

In the above example, the loop is broken due to break statement therefore the else statement will not be executed. The statement present immediate next to else block will be executed.

Output:

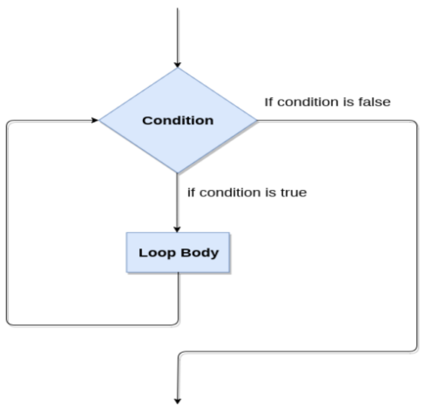
```
0
The loop is broken due to break statement...came out of loop
```

while loop

- The while loop is also known as a pre-tested loop.
- while loop allows a part of the code to be executed as long as the given condition is true.
- It can be viewed as a repeating if statement.
- The while loop is mostly used in the case where the number of iterations is not known in advance.

Syntax is given below.

```
while expression:
   statements
```



Here, the statements can be a single statement or the group of statements. The expression should be any valid python expression resulting into true or false. The true is any non-zero value.

Example 1

```
i=1;
while i<=10:
   print(i);
   i=i+1;
```

Output:

```
1
2
………
```

Example 2

```
i=1
number=0
b=9
number = int(input("Enter the number?"))
while i<=10:
    print("%d X %d = %d \n"%(number,i,number*i));
    i = i+1;
```

Output:

```
Enter the number?10
10 X 1 = 10
```

Infinite while loop:-

If the condition given in the while loop never becomes false then the while loop will never terminate and result into the infinite while loop.

Any non-zero value in the while loop indicates an always-true condition whereas 0 indicates the always-false condition. This type of approach is useful if we want our program to run continuously in the loop without any disturbance.

Example 1

```
while (1):
    print("Hi! we are inside the infinite while loop");
```

Output:

```
Hi! we are inside the infinite while loop
(infinite times)
```

Example 2

```
var = 1
while var != 2:
    i = int(input("Enter the number?"))
    print ("Entered value is %d"%(i))
```

Output:

```
Enter the number?102
Entered value is 102
Enter the number?102
Entered value is 102
Enter the number?103
Entered value is 103
Enter the number?103
(infinite loop)
```

while loop with else:-

Python enables us to use the while loop with the while loop also. The else block is executed when the condition given in the while statement becomes false. Like for loop, if the while loop is broken using break statement, then the else block will not be executed and the statement present after else block will be executed.

Consider the following example.

```
i=1;
while i<=5:
    print(i)
    i=i+1;
else:print("The while loop exhausted");
```

Output:

```
1
2
……..
The while loop exhausted
```

Example 2

```
i=1;
while i<=5:
    print(i)
    i=i+1;
    if(i==3):
        break;
else:print("The while loop exhausted");
```

Output:

```
1
2
```

Break statement:-

The break is a keyword in python which is used to bring the program control out of the loop. The break statement breaks the loops one by one, i.e., in the case of nested loops, it breaks the inner loop first and then proceeds to outer loops. In other words, we can say that break is used to abort the current execution of the program and the control goes to the next line after the loop.

The break is commonly used in the cases where we need to break the loop for a given condition.

The syntax of the break is given below.

```
#loop statements
break;
```

Example 1

```
list =[1,2,3,4]
count = 1;
for i in list:
    if i == 4:
        print("item matched")
        count = count + 1;
        break
print("found at",count,"location");
```

Output:

```
item matched
found at 2 location
```

Example 2

```
str = "python"
for i in str:
    if i == 'o':
        break
    print(i);
```

Output:

```
p
y
t
h
```

Example 3: break statement with while loop

```
i = 0;
while 1:
    print(i," ",end=""),
    i=i+1;
    if i == 10:
        break;
print("came out of while loop");
```

Output:

```
0 1 2 3 4 5 6 7 8 9 came out of while loop
```

Example 3

```
n=2
while 1:
    i=1;
    while i<=10:
        print("%d X %d = %d\n"%(n,i,n*i));
        i = i+1;
    choice = int(input("Do you want to continue printing the table, press 0 for no? "))
    if choice == 0:
        break;
    n=n+1
```

Output:

```
2 X 1 = 2
……….
Do you want to continue printing the table, press 0 for no? 1

3 X 1 = 3
…………
Do you want to continue printing the table, press 0 for no? 0
```

Continue Statement:-

- The continue statement in python is used to bring the program control to the beginning of the loop.
- The continue statement skips the remaining lines of code inside the loop and start with the next iteration.
- It is mainly used for a particular condition inside the loop so that we can skip some specific code for a particular condition.

The syntax of Python continue statement is given below.

```
#loop statements
continue;
#the code to be skipped
```

Example 1

```
i = 0;
while i!=10:
    print("%d"%i);
    continue;
    i=i+1;
```

Output:

infinite loop

Example 2

```
i=1; #initializing a local variable
#starting a loop from 1 to 10
for i in range(1,11):
    if i==5:
        continue;
    print("%d"%i);
```

Output:

```
1
2
……..
```

Pass Statement:-

- The pass statement is a null operation since nothing happens when it is executed.
- It is used in the cases where a statement is syntactically needed but we don't want to use any executable statement at its place.
- For example, it can be used while overriding a parent class method in the subclass but don't want to give its specific implementation in the subclass.
- Pass is also used where the code will be written somewhere but not yet written in the program file.

The syntax of the pass statement is given below.

Example

```
list = [1,2,3,4,5]
flag = 0
for i in list:
    print("Current element:",i,end=" ");
    if i==3:
        pass;
        print("\nWe are inside pass block\n");
        flag = 1;
    if flag==1:
        print("\nCame out of pass\n");
        flag=0;
```

Output:

```
Current element: 1 Current element: 2 Current element: 3
```

```
We are inside pass block
Came out of pass
Current element: 4 Current element: 5
```

List

- Implemented to store the sequence of various type of data.
- Python contains six data types that are capable to store the sequences but the most common and reliable type is list.
- A list can be defined as a collection of values or items of different types.
- The items in the list are separated with the comma (,) and enclosed with the square brackets [].

  Examples:-

```
L1 = ["John", 102, "USA"]
L2 = [1, 2, 3, 4, 5, 6]
L3 = [1, "Ryan"]
```

Example 2:-

```
emp = ["John", 102, "USA"]
Dep1 = ["CS",10];
Dep2 = ["IT",11];
HOD_CS = [10,"Mr. Holding"]
HOD_IT = [11, "Mr. Bewon"]
print("printing employee data...");
print("Name : %s, ID: %d, Country: %s"%(emp[0],emp[1],emp[2]))
print("printing departments...");
print("Department 1:\nName: %s, ID: %d\nDepartment 2:\nName: %s, ID: %s"%(Dep1[0],
Dep2[1],Dep2[0],Dep2[1]));
print("HOD Details ....");
print("CS HOD Name: %s, Id: %d"%(HOD_CS[1],HOD_CS[0]));
print("IT HOD Name: %s, Id: %d"%(HOD_IT[1],HOD_IT[0]));
print(type(emp),type(Dep1),type(Dep2),type(HOD_CS),type(HOD_IT));
```

Output:

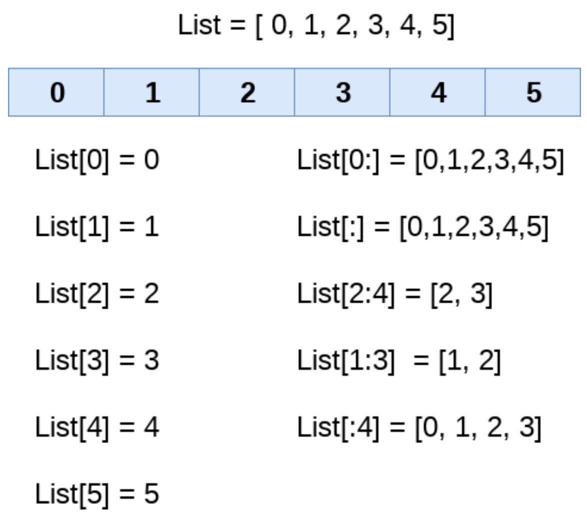
```
printing employee data...
Name : John, ID: 102, Country: USA
printing departments...
Department 1:
Name: CS, ID: 11
Department 2:
Name: IT, ID: 11
HOD Details ....
```

CS HOD Name: Mr. Holding, Id: 10
IT HOD Name: Mr. Bewon, Id: 11
<class 'list'> <class 'list'> <class 'list'> <class 'list'> <class 'list'>

List indexing and splitting

- It is processed in the same way as it happens with the strings.
- The elements of the list can be accessed by using the slice operator [].
- The index starts from 0 and goes to length - 1.
- The first element of the list is stored at the 0th index, the second element of the list is stored at the 1st index, and so on.

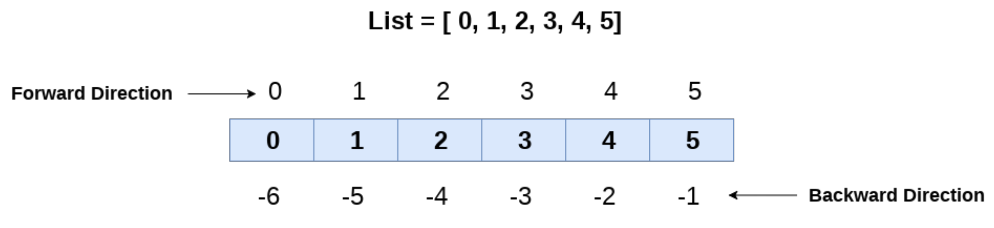
Consider the following example.



```
List = [ 0, 1, 2, 3, 4, 5]

List[0] = 0        List[0:] = [0,1,2,3,4,5]
List[1] = 1        List[:] = [0,1,2,3,4,5]
List[2] = 2        List[2:4] = [2, 3]
List[3] = 3        List[1:3]  = [1, 2]
List[4] = 4        List[:4] = [0, 1, 2, 3]
List[5] = 5
```

- Python provides us the flexibility to use the negative indexing also.
- The negative indices are counted from the right.
- The last element (right most) of the list has the index -1, its adjacent left element is present at the index -2 and so on until the left most element is encountered.



Updating List values

- Lists are the most versatile data structures in python since they are immutable and their values can be updated by using the slice and assignment operator.
- Python also provide us the append() method which can be used to add values to the string.
  Example

```
List = [1, 2, 3, 4, 5, 6]
print(List)
List[2] = 10;
print(List)
List[1:3] = [89, 78]
print(List)
```

Output:

```
[1, 2, 3, 4, 5, 6]
[1, 2, 10, 4, 5, 6]
[1, 89, 78, 4, 5, 6]
```

- The list elements can also be deleted by using the **del** keyword.
- Python also provides us the remove() method if we do not know which element is to be deleted from the list.

  Example

```
List = [0,1,2,3,4]
print(List)
del List[0]
print(List)
del List[3]
print(List)
```

Output:

```
[0, 1, 2, 3, 4]
[1, 2, 3, 4]
[1, 2, 3]
```

Python List Operations

The concatenation (+) and repetition (*) operator work in the same way as they were working with the strings.
Lets see how the list responds to various operators.

1. Consider a List l1 = [1, 2, 3, 4], **and** l2 = [5, 6, 7, 8]

Operator	Description	Example
Repetition	The repetition operator enables the list elements to be repeated multiple times.	L1*2 = [1, 2, 3, 4, 1, 2, 3, 4]
Concatenation	It concatenates the list mentioned on	l1+l2 = [1, 2, 3, 4, 5, 6, 7, 8]

	either side of the operator.	
Membership	It returns true if a particular item exists in a particular list otherwise false.	print(2 in l1) prints True.
Iteration	The for loop is used to iterate over the list elements.	for i in l1: print(i) **Output** 1 . . . . .
Length	It is used to get the length of the list	len(l1) = 4

Iterating a List

A list can be iterated by using a for - in loop. A simple list containing four strings can be iterated as follows.

1. List = ["John", "David", "James", "Jonathan"]
2. **for** i **in** List: #i will iterate over the elements of the List and contains each element in each iter ation.
3.   **print**(i);

Output:

```
John
David
James
Jonathan
```

Adding elements to the list

- Python provides append() function by using which we can add an element to the list. However, the append() method can only add the value to the end of the list.
  Example
  1. l =[];
  2. n = int(input("Enter the number of elements in the list")); #Number of elements will be entered by the user
  3. **for** i **in** range(0,n): # for loop to take the input
  4.   l.append(input("Enter the item?")); # The input is taken from the user and added to t he list as the item
  5. **print**("printing the list items....");
  6. **for** i **in** l: # traversal loop to print the list items

```
    print(i, end = "  ");
```

Output:

```
Enter the number of elements in the list 5
Enter the item?1
Enter the item?2
Enter the item?3
Enter the item?4
Enter the item?5
printing the list items....
  1   2  3  4  5
```

Removing elements from the list

```
List = [0,1,2,3,4]
print("printing original list: ");
for i in List:
   print(i,end=" ")
List.remove(0)
print("\nprinting the list after the removal of first element...")
for i in List:
   print(i,end=" ")
```

Output:

```
printing original list:
0 1 2 3 4
printing the list after the removal of first element...
1 2 3 4
```

Python List Built-in functions

SN	Function	Description
1	cmp(list1, list2)	It compares the elements of both the lists.
2	len(list)	It is used to calculate the length of the list.
3	max(list)	It returns the maximum element of the list.
4	min(list)	It returns the minimum element of the list.

5	list(seq)	It converts any sequence to the list.

List built-in methods

SN	Function	Description
1	list.append(obj)	The element represented by the object obj is added to the list.
2	list.clear()	It removes all the elements from the list.
3	List.copy()	It returns a shallow copy of the list.
4	list.count(obj)	It returns the number of occurrences of the specified object in the list.
5	list.extend(seq)	The sequence represented by the object seq is extended to the list.
6	list.index(obj)	It returns the lowest index in the list that object appears.
7	list.insert(index, obj)	The object is inserted into the list at the specified index.
8	list.pop(obj=list[-1])	It removes and returns the last object of the list.
9	list.remove(obj)	It removes the specified object from the list.
10	list.reverse()	It reverses the list.
11	list.sort([func])	It sorts the list by using the specified compare function if given.

Tuple

- Python Tuple is used to store the sequence of immutable python objects.

- Tuple is similar to lists since the value of the items stored in the list can be changed whereas the tuple is immutable and the value of the items stored in the tuple can not be changed.
- A tuple can be written as the collection of comma-separated values enclosed with the small brackets. A tuple can be defined as follows.

```
T1 = (101, "Ayush", 22)
T2 = ("Apple", "Banana", "Orange")
```

Example 1

```
tuple1 = (10, 20, 30, 40, 50, 60)
print(tuple1)
count = 0
for i in tuple1:
    print("tuple1[%d] = %d"%(count, i));
```

Output:

```
(10, 20, 30, 40, 50, 60)
tuple1[0] = 10
tuple1[0] = 20
tuple1[0] = 30
tuple1[0] = 40
tuple1[0] = 50
tuple1[0] = 60
```

Example 2

```
tuple1 = tuple(input("Enter the tuple elements ..."))
print(tuple1)
count = 0
for i in tuple1:
    print("tuple1[%d] = %s"%(count, i));
```

Output:

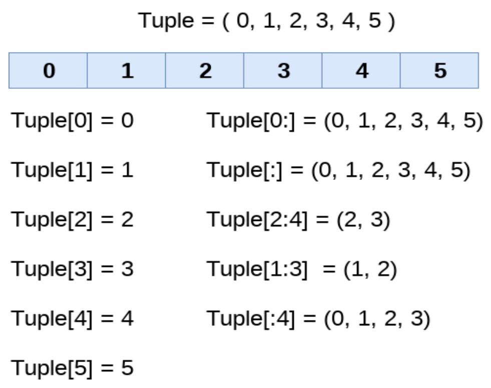
```
Enter the tuple elements ...12345
('1', '2', '3', '4', '5')
tuple1[0] = 1
tuple1[0] = 2
tuple1[0] = 3
tuple1[0] = 4
tuple1[0] = 5
```

- if we try to reassign the items of a tuple, we would get an error as the tuple object doesn't support the item assignment.
  An empty tuple can be written as follows.
  ```
  T3 = ()
  ```
  The tuple having a single value must include a comma as given below.
  ```
  T4 = (90,)
  ```
- A tuple is indexed in the same way as the lists. The items in the tuple can be accessed by using their specific index value.

Tuple indexing and splitting

- The indexing and slicing in tuple are similar to lists. The indexing in the tuple starts from 0 and goes to length(tuple) - 1.
- The items in the tuple can be accessed by using the slice operator. Python also allows us to use the colon operator to access multiple items in the tuple.

Consider the following image to understand the indexing and slicing in detail.



```
Tuple = ( 0, 1, 2, 3, 4, 5 )
```

0	1	2	3	4	5

```
Tuple[0] = 0        Tuple[0:] = (0, 1, 2, 3, 4, 5)
Tuple[1] = 1        Tuple[:] = (0, 1, 2, 3, 4, 5)
Tuple[2] = 2        Tuple[2:4] = (2, 3)
Tuple[3] = 3        Tuple[1:3]  = (1, 2)
Tuple[4] = 4        Tuple[:4] = (0, 1, 2, 3)
Tuple[5] = 5
```

- Unlike lists, the tuple items cannot be deleted by using the del keyword as tuples are immutable. To delete an entire tuple, we can use the del keyword with the tuple name.

Consider the following example.

```
1. tuple1 = (1, 2, 3, 4, 5, 6)
2. print(tuple1)
3. del tuple1[0]
4. print(tuple1)
5. del tuple1
6. print(tuple1)
```

Output:

```
(1, 2, 3, 4, 5, 6)
Traceback (most recent call last):
  File "tuple.py", line 4, in <module>
    print(tuple1)
NameError: name 'tuple1' is not defined
```

- Like lists, the tuple elements can be accessed in both the directions.
- The right most element (last) of the tuple can be accessed by using the index -1.
- The elements from left to right are traversed using the negative indexing.

  Consider the following example.

```
1. tuple1 = (1, 2, 3, 4, 5)
2. print(tuple1[-1])
3. print(tuple1[-4])
```

  Output:

```
5
2
```

Basic Tuple operations

- The operators like concatenation (+), repetition (*), Membership (in) works in the same way as they work with the list. Consider the following table for more detail.
  Let's say Tuple t = (1, 2, 3, 4, 5) and Tuple t1 = (6, 7, 8, 9) are declared.

Operator	**Description**	**Example**
Repetition	The repetition operator enables the tuple elements to be repeated multiple times.	T1*2 = (1, 2, 3, 4, 5, 1, 2, 3, 4, 5)
Concatenation	It concatenates the tuple mentioned on either side of the operator.	T1+T2 = (1, 2, 3, 4, 5, 6, 7, 8, 9)
Membership	It returns true if a particular item exists in the tuple otherwise false.	print (2 in T1) prints True.

Iteration	The for loop is used to iterate over the tuple elements.	for i in T1:
		    print(i)
		Output
		1 . . . . . .

Length	It is used to get the length of the tuple.	len(T1) = 5

Tuple inbuilt functions

SN	**Function**	**Description**
1	cmp(tuple1, tuple2)	It compares two tuples and returns true if tuple1 is greater than tuple2 otherwise false.
2	len(tuple)	It calculates the length of the tuple.
3	max(tuple)	It returns the maximum element of the tuple.
4	min(tuple)	It returns the minimum element of the tuple.
5	tuple(seq)	It converts the specified sequence to the tuple.

Where use tuple
Using tuple instead of list is used in the following scenario.
1. Using tuple instead of list gives us a clear idea that tuple data is constant and must not be

changed.
2. Tuple can simulate dictionary without keys. Consider the following nested structure which can be used as a dictionary.

```
[(101, "John", 22), (102, "Mike", 28),  (103, "Dustin", 30)]
```

3. Tuple can be used as the key inside dictionary due to its immutable nature.

List VS Tuple

SN	List	Tuple
1	The literal syntax of list is shown by the [].	The literal syntax of the tuple is shown by the ().
2	The List is mutable.	The tuple is immutable.
3	The List has the variable length.	The tuple has the fixed length.
4	The list provides more functionality than tuple.	The tuple provides less functionality than the list.
5	The list Is used in the scenario in which we need to store the simple collections with no constraints where the value of the items can be changed.	The tuple is used in the cases where we need to store the read-only collections i.e., the value of the items can not be changed. It can be used as the key inside the dictionary.

Nesting List and tuple:-

We can store list inside tuple or tuple inside the list up to any number of level.
Example of how can we store the tuple inside the list.

```
Employees = [(101, "Ayush", 22), (102, "john", 29), (103, "james", 45), (104, "Ben", 34)]
print("----Printing list----");
for i in Employees:
    print(i)
Employees[0] = (110, "David",22)
print();
print("----Printing list after modification----");
```

8. **for** i **in** Employees:
9.   **print**(i)

Output:
```
----Printing list----
(101, 'Ayush', 22)
(102, 'john', 29)
(103, 'james', 45)
(104, 'Ben', 34)

----Printing list after modification----

(110, 'David', 22)
(102, 'john', 29)
(103, 'james', 45)
(104, 'Ben', 34)
```

Dictionary

- Dictionary is used to implement the key-value pair in python.
- The dictionary is the data type in python which can simulate the real-life data arrangement where some specific value exists for some particular key.
- Dictionary is the collection of key-value pairs where the value can be any python object whereas the keys are the immutable python object, i.e., Numbers, string or tuple.
- Dictionary simulates Java hash-map in python.

Creating the dictionary

The dictionary can be created by using multiple key-value pairs enclosed with the small brackets () and separated by the colon (:). The collections of the key-value pairs are enclosed within the curly braces {}.

The syntax to define the dictionary is given below.

1. Dict = {"Name": "Ayush","Age": 22}

In the above dictionary **Dict**, The keys **Name**, and **Age** are the string that is an immutable object.

Example
1. Employee = {"Name": "John", "Age": 29, "salary":25000,"Company":"GOOGLE"}
2. **print**(type(Employee))
3. **print**("printing Employee data .... ")
4. **print**(Employee)

Output
```
<class 'dict'>
printing Employee data ....
{'Age': 29, 'salary': 25000, 'Name': 'John', 'Company': 'GOOGLE'}
```

Accessing the dictionary values

- The values can be accessed in the dictionary by using the keys as keys are unique in the dictionary.

The dictionary values can be accessed in the following way.

```
Employee = {"Name": "John", "Age": 29, "salary":25000,"Company":"GOOGLE"}
print(type(Employee))
print("printing Employee data .... ")
print("Name : %s" %Employee["Name"])
print("Age : %d" %Employee["Age"])
print("Salary : %d" %Employee["salary"])
print("Company : %s" %Employee["Company"])
```

Output:

```
<class 'dict'>
printing Employee data ....
Name : John
Age : 29
Salary : 25000
Company : GOOGLE
```

- Python provides us with an alternative to use the get() method to access the dictionary values. It would give the same result as given by the indexing.

Updating dictionary values:-

- The dictionary is a mutable data type, and its values can be updated by using the specific keys.

Example

```
Employee = {"Name": "John", "Age": 29, "salary":25000,"Company":"GOOGLE"}
print(type(Employee))
print("printing Employee data .... ")
print(Employee)
print("Enter the details of the new employee....");
Employee["Name"] = input("Name: ");
Employee["Age"] = int(input("Age: "));
Employee["salary"] = int(input("Salary: "));
Employee["Company"] = input("Company:");
print("printing the new data");
print(Employee)
```

Output:

```
<class 'dict'>
```

```
printing Employee data ....
{'Name': 'John', 'salary': 25000, 'Company': 'GOOGLE', 'Age': 29}
Enter the details of the new employee....
Name: David
Age: 19
Salary: 8900
Company:JTP
printing the new data
{'Name': 'David', 'salary': 8900, 'Company': 'JTP', 'Age': 19}
```

Deleting elements using del keyword

The items of the dictionary can be deleted by using the del keyword as given below.

```
Employee = {"Name": "John", "Age": 29, "salary":25000,"Company":"GOOGLE"}
print(type(Employee))
print("printing Employee data .... ")
print(Employee)
print("Deleting some of the employee data")
del Employee["Name"]
del Employee["Company"]
print("printing the modified information ")
print(Employee)
print("Deleting the dictionary: Employee");
del Employee
print("Lets try to print it again ");
print(Employee)
```

Output:

```
<class 'dict'>
printing Employee data ....
{'Age': 29, 'Company': 'GOOGLE', 'Name': 'John', 'salary': 25000}
Deleting some of the employee data
printing the modified information
{'Age': 29, 'salary': 25000}
Deleting the dictionary: Employee
Lets try to print it again
Traceback (most recent call last):
  File "list.py", line 13, in <module>
    print(Employee)
NameError: name 'Employee' is not defined
```

Iterating Dictionary:-

A dictionary can be iterated using the for loop as given below.

Example 1

# for loop to print all the keys of a dictionary

```
Employee = {"Name": "John", "Age": 29, "salary":25000,"Company":"GOOGLE"}
for x in Employee:
    print(x);
```

Output:

```
Name
Company
salary
Age
```

Example 2

#for loop to print all the values of the dictionary

```
Employee = {"Name": "John", "Age": 29, "salary":25000,"Company":"GOOGLE"}
for x in Employee:
    print(Employee[x]);
```

Output:

```
29
GOOGLE
John
25000
```

Example 3

#for loop to print the values of the dictionary by using values() method.

```
Employee = {"Name": "John", "Age": 29, "salary":25000,"Company":"GOOGLE"}
for x in Employee.values():
    print(x);
```

Output:

```
GOOGLE
25000
John
29
```

Example 4

#for loop to print the items of the dictionary by using items() method.

```
Employee = {"Name": "John", "Age": 29, "salary":25000,"Company":"GOOGLE"}
for x in Employee.items():
    print(x);
```

Output:

```
('Name', 'John')
('Age', 29)
('salary', 25000)
('Company', 'GOOGLE')
```

Properties of Dictionary keys

- In the dictionary, we cannot store multiple values for the same keys. If we pass more than one values for a single key, then the value which is last assigned is considered as the value of the key.

  Consider the following example.

  ```
  Employee = {"Name": "John", "Age": 29, "Salary":25000,"Company":"GOOGLE","Name":"Johnn"}
  for x,y in Employee.items():
      print(x,y)
  ```

  Output:

  ```
  Salary 25000
  Company GOOGLE
  Name Johnn
  Age 29
  ```

- The key cannot be any mutable object. We can use numbers, strings, or tuple as the key but we can not use any mutable object like the list as the key in the dictionary.

  Consider the following example.

  ```
  Employee = {"Name": "John", "Age": 29, "salary":25000,"Company":"GOOGLE",[100,201,301]:"Department ID"}
  for x,y in Employee.items():
      print(x,y)
  ```

  Output:

  ```
  Traceback (most recent call last):
   File "list.py", line 1, in
    Employee = {"Name": "John", "Age": 29, "salary":25000,"Company":"GOOGLE",[100,201,301]:"Department ID"}
  TypeError: unhashable type: 'list'
  ```

Built-in Dictionary functions

The built-in python dictionary methods along with the description are given below.

SN	Function	Description
1	cmp(dict1, dict2)	It compares the items of both the dictionary and returns true if the first dictionary values are greater than the second dictionary, otherwise it returns false.

2	len(dict)	It is used to calculate the length of the dictionary.
3	str(dict)	It converts the dictionary into the printable string representation.
4	type(variable)	It is used to print the type of the passed variable.

Built-in Dictionary methods

The built-in python dictionary methods along with the description are given below.

SN	Method	Description
1	dic.clear()	It is used to delete all the items of the dictionary.
2	dict.copy()	It returns a shallow copy of the dictionary.
3	dict.fromkeys(iterable, value = None, /)	Create a new dictionary from the iterable with the values equal to value.
4	dict.get(key, default = "None")	It is used to get the value specified for the passed key.
5	dict.has_key(key)	It returns true if the dictionary contains the specified key.
6	dict.items()	It returns all the key-value pairs as a tuple.
7	dict.keys()	It returns all the keys of the dictionary.
8	dict.setdefault(key,default= "None")	It is used to set the key to the default value if the key is not specified in the dictionary
9	dict.update(dict2)	It updates the dictionary by adding the key-value pair of

		dict2 to this dictionary.
10	dict.values()	It returns all the values of the dictionary.
11	len() , popItem() , pop() , count() , index()	