



DURGA
SOFTWARE SOLUTIONS™

Learn Complete Python In Simple Way



Introduction

- Python is a general purpose high level programming language.
- Python was developed by Guido Van Rossum in 1989 while working at National Research Institute at Netherlands.
- But officially Python was made available to public in 1991. The official Date of Birth for Python is : Feb 20th 1991.
- Python is recommended as first programming language for beginners.

Eg1: To print Helloworld

Java:

```
1) public class HelloWorld  
2) {  
3)     p s v main(String[] args)  
4) {  
5)     SOP("Hello world");  
6) }  
7) }
```

C:

```
1) #include<stdio.h>  
2) void main()  
3) {  
4)     print("Hello world");  
5) }
```

Python:

```
print("Hello World")
```

Eg2: To print the sum of 2 numbers

Java:

```
1) public class Add  
2) {  
3)     public static void main(String[] args)  
4) {
```



```
5) int a,b;
6) a =10;
7) b=20;
8) System.out.println("The Sum:"+ (a+b));
9)
10) }
```

C:

```
1) #include <stdio.h>
2)
3) void main()
4) {
5)     int a,b;
6)     a =10;
7)     b=20;
8)     printf("The Sum:%d", (a+b));
9) }
```

Python:

```
1) a=10
2) b=20
3) print("The Sum:",(a+b))
```

The name Python was selected from the TV Show
"The Complete Monty Python's Circus", which was broadcasted in BBC from 1969 to 1974.

Guido developed Python language by taking almost all programming features from different languages

1. Functional Programming Features from C
2. Object Oriented Programming Features from C++
3. Scripting Language Features from Perl and Shell Script
4. Modular Programming Features from Modula-3

Most of syntax in Python Derived from C and ABC languages.

Where we can use Python:

We can use everywhere. The most common important application areas are

- 1) For developing Desktop Applications
- 2) For developing web Applications
- 3) For developing database Applications



- 4) For Network Programming
 - 5) For developing games
 - 6) For Data Analysis Applications
 - 7) For Machine Learning
 - 8) For developing Artificial Intelligence Applications
 - 9) For IoT
-

Note:

- Internally Google and Youtube use Python coding.
- NASA and Nework Stock Exchange Applications developed by Python.
- Top Software companies like Google, Microsoft, IBM, Yahoo using Python.

Features of Python:

1) Simple and easy to learn:

- Python is a simple programming language. When we read Python program,we can feel like reading english statements.
- The syntaxes are very simple and only 30+ keywords are available.
- When compared with other languages, we can write programs with very less number of lines. Hence more readability and simplicity.
- We can reduce development and cost of the project.

2) Freeware and Open Source:

- We can use Python software without any licence and it is freeware.
- Its source code is open,so that we can we can customize based on our requirement.
- Eg: Jython is customized version of Python to work with Java Applications.

3) High Level Programming language:

- Python is high level programming language and hence it is programmer friendly language.
- Being a programmer we are not required to concentrate low level activities like memory management and security etc.

4) Platform Independent:

- Once we write a Python program,it can run on any platform without rewriting once again.
- Internally PVM is responsible to convert into machine understandable form.

5) Portability:

Python programs are portable. ie we can migrate from one platform to another platform very easily. Python programs will provide same results on any paltform.



6) Dynamically Typed:

- In Python we are not required to declare type for variables. Whenever we are assigning the value, based on value, type will be allocated automatically. Hence Python is considered as dynamically typed language.
- But Java, C etc are Statically Typed Languages b'z we have to provide type at the beginning only.
- This dynamic typing nature will provide more flexibility to the programmer.

7) Both Procedure Oriented and Object Oriented:

Python language supports both Procedure oriented (like C, pascal etc) and object oriented (like C++, Java) features. Hence we can get benefits of both like security and reusability etc

8) Interpreted:

- We are not required to compile Python programs explicitly. Internally Python interpreter will take care that compilation.
- If compilation fails interpreter raised syntax errors. Once compilation success then PVM (Python Virtual Machine) is responsible to execute.

9) Extensible:

- We can use other language programs in Python.
- The main advantages of this approach are:
 - We can use already existing legacy non-Python code
 - We can improve performance of the application

10) Embedded:

We can use Python programs in any other language programs.
i.e we can embedd Python programs anywhere.

11) Extensive Library:

- Python has a rich inbuilt library.
- Being a programmer we can use this library directly and we are not responsible to implement the functionality. Etc.

Limitations of Python:

- 1) Performance wise not up to the mark because it is interpreted language.
- 2) Not using for mobile Applications.



Flavors of Python:

1) CPython:

It is the standard flavor of Python. It can be used to work with C language Applications.

2) Jython OR JPython:

It is for Java Applications. It can run on JVM

3) IronPython:

It is for C#.Net platform

4) PyPy:

The main advantage of PyPy is performance will be improved because JIT compiler is available inside PVM.

5) RubyPython

For Ruby Platforms

6) AnacondaPython

It is specially designed for handling large volume of data processing.

Python Versions:

- Python 1.0V introduced in Jan 1994
- Python 2.0V introduced in October 2000
- Python 3.0V introduced in December 2008

Note: Python 3 won't provide backward compatibility to Python2 i.e there is no guarantee that Python2 programs will run in Python3.

Current versions

Python 3.6.1

Python 2.7.13



IDENTIFIERS

- A Name in Python Program is called Identifier.
- It can be Class Name OR Function Name OR Module Name OR Variable Name.
- `a = 10`

Rules to define Identifiers in Python:

1. The only allowed characters in Python are

- alphabet symbols(either lower case or upper case)
- digits(0 to 9)
- underscore symbol(_)

By mistake if we are using any other symbol like \$ then we will get syntax error.

- `cash = 10` ✓
- `ca$h =20` X

2. Identifier should not starts with digit

- `123total` X
- `total123` ✓

3. Identifiers are case sensitive. Of course Python language is case sensitive language.

- `total=10`
- `TOTAL=999`
- `print(total) #10`
- `print(TOTAL) #999`



Identifier:

- 1) Alphabet Symbols (Either Upper case OR Lower case)
- 2) If Identifier is start with Underscore (_) then it indicates it is private.
- 3) Identifier should not start with Digits.
- 4) Identifiers are case sensitive.
- 5) We cannot use reserved words as identifiers
Eg: def = 10 X
- 6) There is no length limit for Python identifiers. But not recommended to use too lengthy identifiers.
- 7) Dollar (\$) Symbol is not allowed in Python.

Q) Which of the following are valid Python identifiers?

- 1) 123total X
- 2) total123 ✓
- 3) java2share ✓
- 4) ca\$h X
- 5) _abc_abc_ ✓
- 6) def X
- 7) if X

Note:

- 1) If identifier starts with _ symbol then it indicates that it is private
- 2) If identifier starts with __(Two Under Score Symbols) indicating that strongly private identifier.
- 3) If the identifier starts and ends with two underscore symbols then the identifier is language defined special name, which is also known as magic methods.
- 4) Eg: __add__



RESERVED WORDS

In Python some words are reserved to represent some meaning or functionality. Such types of words are called reserved words.

There are 33 reserved words available in Python.

- True, False, None
- and, or ,not,is
- if, elif, else
- while, for, break, continue, return, in, yield
- try, except, finally, raise, assert
- import, from, as, class, def, pass, global, nonlocal, lambda, del, with

Note:

1. All Reserved words in Python contain only alphabet symbols.
2. Except the following 3 reserved words, all contain only lower case alphabet symbols.

- True
- False
- None

Eg: a=true **X**
a=True **J**

```
>>> import keyword  
>>> keyword.kwlist  
['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else',  
'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or',  
'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
```

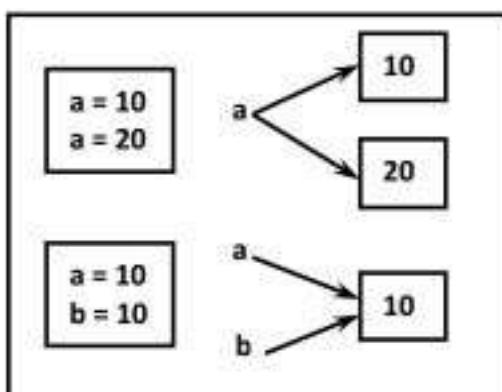


DATA TYPES

- Data Type represents the type of data present inside a variable.
- In Python we are not required to specify the type explicitly. Based on value provided, the type will be assigned automatically. Hence Python is dynamically Typed Language.

Python contains the following inbuilt data types

- 1) Int
- 2) Float
- 3) Complex
- 4) Bool
- 5) Str
- 6) Bytes
- 7) Bytearray
- 8) Range
- 9) List
- 10) Tuple
- 11) Set
- 12) Frozenset
- 13) Dict
- 14) None



Note: Python contains several inbuilt functions

1) type()

to check the type of variable

2) id()

to get address of object



3) print()

to print the value

In Python everything is an Object.

1) int Data Type:

We can use int data type to represent whole numbers (integral values)

Eg: a = 10
type(a) #int

Note:

- In Python2 we have long data type to represent very large integral values.
- But in Python3 there is no long type explicitly and we can represent long values also by using int type only.

We can represent int values in the following ways

- 1) Decimal form
- 2) Binary form
- 3) Octal form
- 4) Hexa decimal form

I) Decimal Form (Base-10):

- It is the default number system in Python
- The allowed digits are: 0 to 9
- Eg: a = 10

II) Binary Form (Base-2):

- The allowed digits are : 0 & 1
- Literal value should be prefixed with 0b or 0B
- Eg: a = 0B1111
a = 0B123
a = b111



III) Octal Form (Base-8):

- The allowed digits are : 0 to 7
- Literal value should be prefixed with 0o or 0O.
- Eg: a = 0o123
a = 0o786

IV) Hexa Decimal Form (Base-16):

- The allowed digits are: 0 to 9, a-f (both lower and upper cases are allowed)
- Literal value should be prefixed with 0x or 0X
- Eg: a = 0XFACE
a = 0XBeef
a = 0XBeer

Note: Being a programmer we can specify literal values in decimal, binary, octal and hexa decimal forms. But PVM will always provide values only in decimal form.

- a=10
- b=0o10
- c=0X10
- d=0B10
- print(a)10
- print(b)8
- print(c)16
- print(d)2

Base Conversions

Python provide the following in-built functions for base conversions

1) bin():

We can use bin() to convert from any base to binary

- 1) >>> bin(15)
- 2) '0b1111'
- 3) >>> bin(0o11)
- 4) '0b1001'
- 5) >>> bin(0X10)



6) '0b10000'

2) oct():

We can use oct() to convert from any base to octal

- 1) >>> oct(10)
- 2) '0o12'
- 3) >>> oct(0B11111)
- 4) '0o17'
- 5) >>> oct(0X123)
- 6) '0o443'

3) hex():

We can use hex() to convert from any base to hexa decimal

- 1) >>> hex(100)
- 2) '0x64'
- 3) >>> hex(0B111111)
- 4) '0x3f'
- 5) >>> hex(0o12345)
- 6) '0x14e5'

2) Float Data Type:

- We can use float data type to represent floating point values (decimal values)
Eg: f = 1.234
type(f) float
- We can also represent floating point values by using exponential form (Scientific Notation)
Eg: f = 1.2e3 → instead of 'e' we can use 'E'
print(f) 1200.0
- The main advantage of exponential form is we can represent big values in less memory.

***Note:

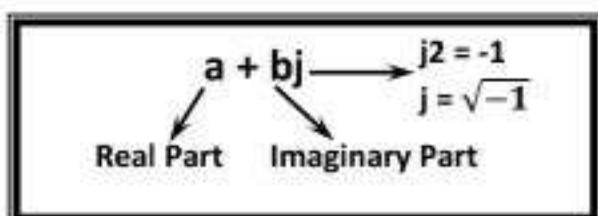
We can represent int values in decimal, binary, octal and hexa decimal forms. But we can represent float values only by using decimal form.



```
1) >>> f=0B11.01
2) File "<stdin>", line 1
3)   f=0B11.01
4)     ^
5) SyntaxError: invalid syntax
6)
7) >>> f=0o123.456
8) SyntaxError: invalid syntax
9)
10) >>> f=0X123.456
11) SyntaxError: invalid syntax
```

3) Complex Data Type:

- A complex number is of the form



- 'a' and 'b' contain Integers OR Floating Point Values.

Eg: $3 + 5j$

$10 + 5.5j$

$0.5 + 0.1j$

- In the real part if we use int value then we can specify that either by decimal, octal, binary or hexa decimal form.
- But imaginary part should be specified only by using decimal form.

```
1) >>> a=0B11+5j
2) >>> a
3) (3+5j)
4) >>> a=3+0B11j
5) SyntaxError: invalid syntax
```

- Even we can perform operations on complex type values.

```
1) >>> a=10+1.5j
2) >>> b=20+2.5j
3) >>> c=a+b
4) >>> print(c)
```



- 5) (30+4j)
- 6) >>> type(c)
- 7) <class 'complex'>

Note: Complex data type has some inbuilt attributes to retrieve the real part and imaginary part

c = 10.5+3.6j

c.real → 10.5
c.imag → 3.6

We can use complex type generally in scientific Applications and electrical engineering Applications.

4) bool Data Type:

- We can use this data type to represent boolean values.
- The only allowed values for this data type are:
- True and False
- Internally Python represents True as 1 and False as 0

b = True
type(b) → bool

Eg:

a = 10
b = 20
c = a<b
print(c) → True

True+True → 2
True-False → 1

5) str Data Type:

- str represents String data type.
- A String is a sequence of characters enclosed within single quotes or double quotes.



- s1='durga'
- s1="durga"
- By using single quotes or double quotes we cannot represent multi line string literals.
- s1="durga
soft"
- For this requirement we should go for triple single quotes('') or triple double quotes("")
- s1="""durga
soft"""
- s1=""""durga
soft"""
- We can also use triple quotes to use single quote or double quote in our String.
- """ This is " character""
' This i " Character '
- We can embed one string in another string
- """This "Python class very helpful" for java students""

Slicing of Strings:

- 1) slice means a piece
- 2) [] operator is called slice operator, which can be used to retrieve parts of String.
- 3) In Python Strings follows zero based index.
- 4) The index can be either +ve or -ve.
- 5) +ve index means forward direction from Left to Right
- 6) -ve index means backward direction from Right to Left

	-5	-4	-3	-2	-1
	d	u	r	g	a
	0	1	2	3	4

- 1) >>> s="durga"
- 2) >>> s[0]
- 3) 'd'
- 4) >>> s[1]
- 5) 'u'
- 6) >>> s[-1]
- 7) 'a'
- 8) >>> s[40]



IndexError: string index out of range

```
1) >>> s[1:40]
2) 'urga'
3) >>> s[1:]
4) 'urga'
5) >>> s[:4]
6) 'durg'
7) >>> s[:]
8) 'durga'
9) >>>
10)
11)>>> s*3
12)'durgadurgadurga'
13)
14)>>> len(s)
15) 5
```

Note:

- 1) In Python the following data types are considered as Fundamental Data types
 - int
 - float
 - complex
 - bool
 - str
- 2) In Python, we can represent char values also by using str type and explicitly char type is not available.

```
1) >>> c='a'
2) >>> type(c)
3) <class 'str'>
```
- 3) long Data Type is available in Python2 but not in Python3. In Python3 long values also we can represent by using int type only.
- 4) In Python we can present char Value also by using str Type and explicitly char Type is not available.



TYPE CASTING

- ⦿ We can convert one type value to another type. This conversion is called Typecasting or Type coercion.
- ⦿ The following are various inbuilt functions for type casting.

- 1) int()
- 2) float()
- 3) complex()
- 4) bool()
- 5) str()

int():

We can use this function to convert values from other types to int

```
1) >>> int(123.987)
2) 123
3) >>> int(10+5j)
4) TypeError: can't convert complex to int
5) >>> int(True)
6) 1
7) >>> int(False)
8) 0
9) >>> int("10")
10) 10
11) >>> int("10.5")
12) ValueError: invalid literal for int() with base 10: '10.5'
13) >>> int("ten")
14) ValueError: invalid literal for int() with base 10: 'ten'
15) >>> int("OB1111")
16) ValueError: invalid literal for int() with base 10: 'OB1111'
```

Note:

- 1) We can convert from any type to int except complex type.
- 2) If we want to convert str type to int type, compulsory str should contain only integral value and should be specified in base-10.



❖ float():

We can use float() function to convert other type values to float type.

```
1) >>> float(10)
2) 10.0
3) >>> float(10+5j)
4) TypeError: can't convert complex to float
5) >>> float(True)
6) 1.0
7) >>> float(False)
8) 0.0
9) >>> float("10")
10) 10.0
11) >>> float("10.5")
12) 10.5
13) >>> float("ten")
14) ValueError: could not convert string to float: 'ten'
15) >>> float("0B1111")
16) ValueError: could not convert string to float: '0B1111'
```

Note:

- 1) We can convert any type value to float type except complex type.
- 2) Whenever we are trying to convert str type to float type compulsory str should be either integral or floating point literal and should be specified only in base-10.

❖ complex():

We can use complex() function to convert other types to complex type.

Form-1: complex(x)

We can use this function to convert x into complex number with real part x and imaginary part 0.

Eg:

```
1) complex(10)==>10+0j
2) complex(10.5)==>10.5+0j
3) complex(True)==>1+0j
4) complex(False)==>0j
5) complex("10")=>10+0j
6) complex("10.5")=>10.5+0j
7) complex("ten")
8) ValueError: complex() arg is a malformed string
```



Form-2: complex(x,y)

We can use this method to convert x and y into complex number such that x will be real part and y will be imaginary part.

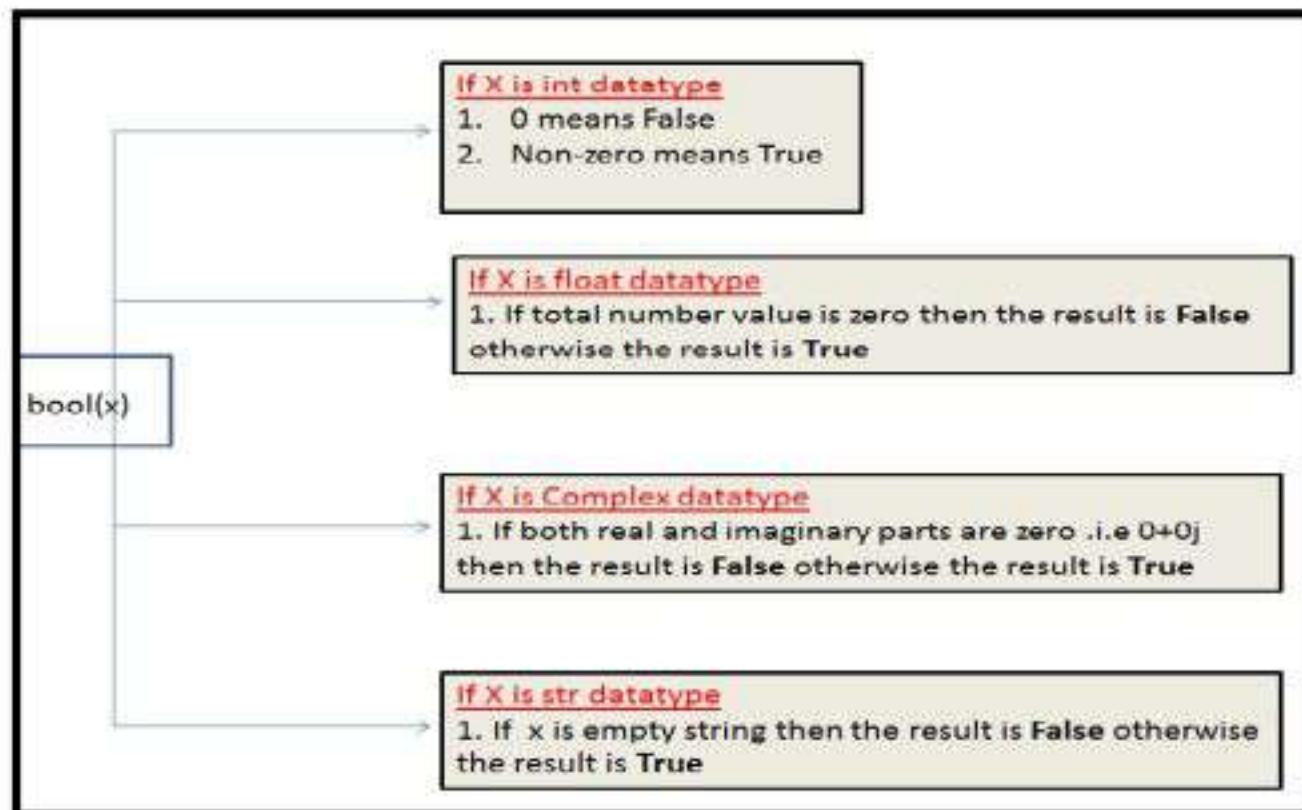
Eg: `complex(10, -2) → 10-2j`
`complex(True, False) → 1+0j`



bool():

We can use this function to convert other type values to bool type.

- 1) `bool(0) → False`
- 2) `bool(1) → True`
- 3) `bool(10) → True`
- 4) `bool(10.5) → True`
- 5) `bool(0.178) → True`
- 6) `bool(0.0) → False`
- 7) `bool(10-2j) → True`
- 8) `bool(0+1.5j) → True`
- 9) `bool(0+0j) → False`
- 10) `bool("True") → True`
- 11) `bool("False") → True`
- 12) `bool("") → False`





⌚ str():

We can use this method to convert other type values to str type.

```
1) >>> str(10)
2) '10'
3) >>> str(10.5)
4) '10.5'
5) >>> str(10+5j)
6) '(10+5j)'
7) >>> str(True)
8) 'True'
```

Fundamental Data Types vs Immutability:

- ⌚ All Fundamental Data types are immutable. i.e once we creates an object,we cannot perform any changes in that object. If we are trying to change then with those changes a new object will be created. This non-changeable behaviour is called immutability.
- ⌚ In Python if a new object is required, then PVM won't create object immediately. First it will check is any object available with the required content or not. If available then existing object will be reused. If it is not available then only a new object will be created. The advantage of this approach is memory utilization and performance will be improved.
- ⌚ But the problem in this approach is, several references pointing to the same object, by using one reference if we are allowed to change the content in the existing object then the remaining references will be effected. To prevent this immutability concept is required. According to this once creates an object we are not allowed to change content. If we are trying to change with those changes a new object will be created.

```
1) >>> a=10
2) >>> b=10
3) >>> a is b
4) True
5) >>> id(a)
6) 1572353952
7) >>> id(b)
8) 1572353952
9) >>>
```



```
>>> a=10  
  
>>> b=10  
  
>>> id(a)  
1572353952  
  
>>> id(b)  
1572353952  
  
>>> a is b  
True
```

```
>>> a=10+5j  
  
>>> b=10+5j  
  
>>> a is b  
False  
  
>>> id(a)  
15980256  
  
>>> id(b)  
15979944
```

```
>>> a=True  
  
>>> b=True  
  
>>> a is b  
True  
  
>>> id(a)  
1572172624  
  
>>> id(b)  
1572172624
```

```
>>> a='durga'  
  
>>> b='durga'  
  
>>> a is b  
True  
  
>>> id(a)  
16378848  
  
>>> id(b)  
16378848
```

6) bytes Data Type:

bytes data type represents a group of byte numbers just like an array.

```
1) x = [10,20,30,40]  
2)   b = bytes(x)  
3)   type(b) → bytes  
4)   print(b[0]) → 10  
5)   print(b[-1]) → 40  
6)   >>> for i in b : print(i)  
7)  
8)     10  
9)     20  
10)    30  
11)    40
```

Conclusion 1:

The only allowed values for byte data type are 0 to 256. By mistake if we are trying to provide any other values then we will get value error.

Conclusion 2:

Once we creates bytes data type value, we cannot change its values, otherwise we will get TypeError.



Eg:

- 1) `>>> x=[10,20,30,40]`
- 2) `>>> b=bytes(x)`
- 3) `>>> b[0]=100`
- 4) `TypeError: 'bytes' object does not support item assignment`

7) bytearray Data Type:

bytearray is exactly same as bytes data type except that its elements can be modified.

Eg.1:

- 1) `x=[10,20,30,40]`
- 2) `b = bytearray(x)`
- 3) `for i in b : print(i)`
- 4) `10`
- 5) `20`
- 6) `30`
- 7) `40`
- 8) `b[0]=100`
- 9) `for i in b: print(i)`
- 10) `100`
- 11) `20`
- 12) `30`
- 13) `40`

Eg.2:

- 1) `>>> x =[10,256]`
- 2) `>>> b = bytearray(x)`
- 3) `ValueError: byte must be in range(0, 256)`

8) List Data Type:

If we want to represent a group of values as a single entity where insertion order required to preserve and duplicates are allowed then we should go for list data type.

- 1) Insertion Order is preserved
- 2) Heterogeneous Objects are allowed
- 3) Duplicates are allowed
- 4) Growable in nature
- 5) Values should be enclosed within square brackets.



Eg:

```
1) list=[10,10.5,'durga',True,10]
2) print(list) # [10,10.5,'durga',True,10]
```

Eg:

```
1) list=[10,20,30,40]
2) >>> list[0]
3) 10
4) >>> list[-1]
5) 40
6) >>> list[1:3]
7) [20, 30]
8) >>> list[0]=100
9) >>> for i in list:print(i)
10) ...
11) 100
12) 20
13) 30
14) 40
```

list is growable in nature. i.e based on our requirement we can increase or decrease the size.

```
1) >>> list=[10,20,30]
2) >>> list.append("durga")
3) >>> list
4) [10, 20, 30, 'durga']
5) >>> list.remove(20)
6) >>> list
7) [10, 30, 'durga']
8) >>> list2=list*2
9) >>> list2
10) [10, 30, 'durga', 10, 30, 'durga']
```

Note: An ordered, mutable, heterogenous collection of elements is nothing but list, where duplicates also allowed.



9) Tuple Data Type:

- tuple data type is exactly same as list data type except that it is immutable.i.e we cannot change values.
- Tuple elements can be represented within parenthesis.

Eg:

```
1) t=(10,20,30,40)
2) type(t)
3) <class 'tuple'>
4) t[0]=100
5) TypeError: 'tuple' object does not support item assignment
6) >>> t.append("durga")
7) AttributeError: 'tuple' object has no attribute 'append'
8) >>> t.remove(10)
9) AttributeError: 'tuple' object has no attribute 'remove'
```

Note: tuple is the read only version of list

10) Range Data Type:

- range Data Type represents a sequence of numbers.
- The elements present in range Data type are not modifiable. i.e range Data type is immutable.

Form-1: range(10)

generate numbers from 0 to 9

Eg:

```
r = range(10)
for i in r : print(i) → 0 to 9
```

Form-2: range(10, 20)

generate numbers from 10 to 19

Eg:

```
r = range(10,20)
for i in r : print(i) → 10 to 19
```



Form-3: range(10, 20, 2)

2 means increment value

Eg:

```
r = range(10,20,2)  
for i in r : print(i) → 10,12,14,16,18
```

We can access elements present in the range Data Type by using index.

Eg:

```
r = range(10,20)  
r[0] → 10  
r[15] → IndexError: range object index out of range
```

We cannot modify the values of range data type

Eg:

```
r[0] = 100  
TypeError: 'range' object does not support item assignment
```

We can create a list of values with range data type

Eg:

- 1) >>> l = list(range(10))
- 2) >>> l
- 3) [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

11) set Data Type:

⦿ If we want to represent a group of values without duplicates where order is not important then we should go for set Data Type.

- 1) Insertion order is not preserved
- 2) Duplicates are not allowed
- 3) Heterogeneous objects are allowed
- 4) Index concept is not applicable
- 5) It is mutable collection
- 6) Growable in nature



Eg:

- 1) s={100,0,10,200,10,'durga'}
- 2) s # {0, 100, 'durga', 200, 10}
- 3) s[0] → TypeError: 'set' object does not support indexing

⦿ set is growable in nature, based on our requirement we can increase or decrease the size.

- 1) >>> s.add(60)
- 2) >>> s
- 3) {0, 100, 'durga', 200, 10, 60}
- 4) >>> s.remove(100)
- 5) >>> s
- 6) {0, 'durga', 200, 10, 60}

12) frozenset Data Type:

⦿ It is exactly same as set except that it is immutable.
⦿ Hence we cannot use add or remove functions.

- 1) >>> s={10,20,30,40}
- 2) >>> fs=frozenset(s)
- 3) >>> type(fs)
- 4) <class 'frozenset'>
- 5) >>> fs
- 6) frozenset({40, 10, 20, 30})
- 7) >>> for i in fs:print(i)
- 8) ...
- 9) 40
- 10) 10
- 11) 20
- 12) 30
- 13)
- 14) >>> fs.add(70)
- 15) AttributeError: 'frozenset' object has no attribute 'add'
- 16) >>> fs.remove(10)
- 17) AttributeError: 'frozenset' object has no attribute 'remove'



13) dict Data Type:

- ⦿ If we want to represent a group of values as key-value pairs then we should go for dict data type.
- ⦿ Eg: `d = {101:'durga',102:'ravi',103:'shiva'}`
- ⦿ Duplicate keys are not allowed but values can be duplicated. If we are trying to insert an entry with duplicate key then old value will be replaced with new value.

Eg:

```
1) >>> d={101:'durga',102:'ravi',103:'shiva'}
2) >>> d[101]='sunny'
3) >>> d
4) {101: 'sunny', 102: 'ravi', 103: 'shiva'}
5)
6) We can create empty dictionary as follows
7) d={}
8) We can add key-value pairs as follows
9) d['a']='apple'
10) d['b']='banana'
11) print(d)
```

Note: dict is mutable and the order won't be preserved.

Note:

- 1) In general we can use bytes and bytearray data types to represent binary information like images, video files etc
- 2) In Python2 long data type is available. But in Python3 it is not available and we can represent long values also by using int type only.
- 3) In Python there is no char data type. Hence we can represent char values also by using str type.



Summary of Datatypes in Python 3

Datatype	Description	Is Immutable?	Example
Int	We can use to represent the whole/integral numbers	Immutable	<pre>>>> a=10 >>> type(a) <class 'int'></pre>
Float	We can use to represent the decimal/floating point numbers	Immutable	<pre>>>> b=10.5 >>> type(b) <class 'float'></pre>
Complex	We can use to represent the complex numbers	Immutable	<pre>>>> c=10+5j >>> type(c) <class 'complex'> >>> c.real 10.0 >>> c.imag 5.0</pre>
Bool	We can use to represent the logical values (Only allowed values are True and False)	Immutable	<pre>>>> flag=True >>> flag=False >>> type(flag) <class 'bool'></pre>
Str	To represent sequence of Characters	Immutable	<pre>>>> s='durga' >>> type(s) <class 'str'> >>> s="durga" >>> s=="Durga Software Solutions... Ameerpet" >>> type(s) <class 'str'></pre>
bytes	To represent a sequence of byte values from 0-255	Immutable	<pre>>>> list=[1,2,3,4] >>> b=bytes(list) >>> type(b) <class 'bytes'></pre>
bytearray	To represent a sequence of byte values from 0-255	Mutable	<pre>>>> list=[10,20,30] >>> ba=bytearray(list) >>> type(ba) <class 'bytearray'></pre>
range	To represent a range of values	Immutable	<pre>>>> r=range(10) >>> r1=range(0,10) >>> r2=range(0,10,2)</pre>



list	To represent an ordered collection of objects	Mutable	>>> l=[10,11,12,13,14,15] >>> type(l) <class 'list'>
tuple	To represent an ordered collections of objects	Immutable	>>> t=(1,2,3,4,5) >>> type(t) <class 'tuple'>
set	To represent an unordered collection of unique objects	Mutable	>>> s={1,2,3,4,5,6} >>> type(s) <class 'set'>
frozenset	To represent an unordered collection of unique objects	Immutable	>>> s={11,2,3,'Durga',100,'Ramu'} >>> fs=frozenset(s) >>> type(fs) <class 'frozenset'>
dict	To represent a group of key value pairs	Mutable	>>> d = {101:'durga', 102:'ramu', 103:'hari'} >>> type(d) <class 'dict'>

14) None Data Type:

- None means nothing or No value associated.
- If the value is not available, then to handle such type of cases None introduced.
- It is something like null value in Java.

Eg:

```
def m1():
    a=10
```

```
print(m1())
None
```



Escape Characters:

In String literals we can use escape characters to associate a special meaning.

```
1) >>> s="durga\nsoftware"
2) >>> print(s)
3) durga
4) software
5) >>> s="durga\tsoftware"
6) >>> print(s)
7) durga  software
8) >>> s="This is " symbol"
9) File "<stdin>", line 1
10) s="This is " symbol"
11) ^
12) SyntaxError: invalid syntax
13)>>> s="This is \" symbol"
14)>>> print(s)
15) This is " symbol
```

The following are various important escape characters in Python

- 1) \n → New Line
- 2) \t → Horizontal Tab
- 3) \r → Carriage Return
- 4) \b → Back Space
- 5) \f → Form Feed
- 6) \v → Vertical Tab
- 7) \' → Single Quote
- 8) \" → Double Quote
- 9) \\ → Back Slash Symbol

....

Constants:

- Constants concept is not applicable in Python.
- But it is convention to use only uppercase characters if we don't want to change value.
- MAX_VALUE = 10
- It is just convention but we can change the value.



- Operator is a symbol that performs certain operations.
- Python provides the following set of operators

- 1) Arithmetic Operators
- 2) Relational Operators OR Comparison Operators
- 3) Logical operators
- 4) Bitwise operators
- 5) Assignment operators
- 6) Special operators

1) Arithmetic Operators:

- 1) + → Addition
- 2) - → Subtraction
- 3) * → Multiplication
- 4) / → Division Operator
- 5) % → Modulo Operator
- 6) // → Floor Division Operator
- 7) ** → Exponent Operator OR Power Operator

Eg: test.py

```
1) a=10
2) b=2
3) print('a+b=',a+b)
4) print('a-b=',a-b)
5) print('a*b=',a*b)
6) print('a/b=',a/b)
7) print('a//b=',a//b)
8) print('a%b=',a%b)
9) print('a**b=',a**b)
```

Output:

Python test.py OR py test.py
a+b = 12
a-b = 8
a*b = 20



a/b= 5.0
a//b= 5
a%b= 0
a**b= 100

Eg:

- 1) a = 10.5
- 2) b=2
- 3)
- 4) a+b= 12.5
- 5) a-b= 8.5
- 6) a*b= 21.0
- 7) a/b= 5.25
- 8) a//b= 5.0
- 9) a%b= 0.5
- 10) a**b= 110.25

Eg:

10/2 → 5.0
10//2 → 5
10.0/2 → 5.0
10.0//2 → 5.0

Note:

- ⦿ / operator always performs floating point arithmetic. Hence it will always returns float value.
- ⦿ But Floor division (//) can perform both floating point and integral arithmetic. If arguments are int type then result is int type. If atleast one argument is float type then result is float type.

Note:

- ⦿ We can use +,* operators for str type also.
- ⦿ If we want to use + operator for str type then compulsory both arguments should be str type only otherwise we will get error.

- 1) >>> "durga"+10
- 2) TypeError: must be str, not int
- 3) >>> "durga"+"10"
- 4) 'durga10'



- ④ If we use * operator for str type then compulsory one argument should be int and other argument should be str type.

- ④ 2*"durga"

"durga"*2

2.5*"durga" → TypeError: can't multiply sequence by non-int of type 'float'

"durga""durga" → TypeError: can't multiply sequence by non-int of type 'str'

- ④ + → String Concatenation Operator

- ④ * → String Multiplication Operator

Note: For any number x,

x/0 and x%0 always raises "ZeroDivisionError"

10/0

10.0/0

....

2) Relational Operators: >, >=, <, <=

```
1) a=10
2) b=20
3) print("a > b is ",a>b)
4) print("a >= b is ",a>=b)
5) print("a < b is ",a<b)
6) print("a <= b is ",a<=b)
7)
8) a > b is False
9) a >= b is False
10) a < b is True
11) a <= b is True
```

We can apply relational operators for str types also.

Eg 2:

```
1) a="durga"
2) b="durga"
3) print("a > b is ",a>b)
4) print("a >= b is ",a>=b)
5) print("a < b is ",a<b)
6) print("a <= b is ",a<=b)
7)
```



- 8) `a > b is False`
- 9) `a >= b is True`
- 10) `a < b is False`
- 11) `a <= b is True`

Eg:

- 1) `print(True>True)` False
- 2) `print(True>=True)` True
- 3) `print(10 >True)` True
- 4) `print(False > True)` False
- 5)
- 6) `print(10>'durga')`
- 7) `TypeError: '>' not supported between instances of 'int' and 'str'`

Eg:

- 1) `a=10`
- 2) `b=20`
- 3) `if(a>b):`
- 4) `print("a is greater than b")`
- 5) `else:`
- 6) `print("a is not greater than b")`

Output: a is not greater than b

Note: Chaining of relational operators is possible. In the chaining, if all comparisons returns True then only result is True. If atleast one comparison returns False then the result is False

- 1) `10<20 → True`
- 2) `10<20<30 → True`
- 3) `10<20<30<40 → True`
- 4) `10<20<30<40>50 → False`

3) Equality Operators: ==, !=

We can apply these operators for any type even for incompatible types also.

- 1) `>>> 10==20`
- 2) `False`
- 3) `>>> 10!= 20`
- 4) `True`
- 5) `>>> 10==True`



- 6) False
- 7) >>> False==False
- 8) True
- 9) >>> "durga"=="durga"
- 10) True
- 11) >>> 10=="durga"
- 12) False

Note: Chaining concept is applicable for equality operators. If atleast one comparison returns False then the result is False. Otherwise the result is True.

- 1) >>> 10==20==30==40
- 2) False
- 3) >>> 10==10==10==10
- 4) True

4) Logical Operators: and, or, not

We can apply for all types.

❖ For boolean Types Behaviour:

and → If both arguments are True then only result is True
or → If atleast one arugemnt is True then result is True
not → Complement

True and False → False

True or False → True

not False → True

❖ For non-boolean Types Behaviour:

0 means False

non-zero means True

empty string is always treated as False

x and y:

If x is evaluates to false return x otherwise return y

Eg:

10 and 20

0 and 20

If first argument is zero then result is zero otherwise result is y



x Or y:

If x evaluates to True then result is x otherwise result is y

10 or 20 → 10

0 or 20 → 20

not x:

If x is evaluated to False then result is True otherwise False

not 10 → False

not 0 → True

Eg:

- 1) "durga" and "durgasoft" ==>durgasoft
- 2) "" and "durga" ==>""
- 3) "durga" and "" ==>""
- 4) "" or "durga" ==>"durga"
- 5) "durga" or ""==>"durga"
- 6) not ""==>True
- 7) not "durga" ==>False

5) Bitwise Operators:

- ⦿ We can apply these operators bitwise.
- ⦿ These operators are applicable only for int and boolean types.
- ⦿ By mistake if we are trying to apply for any other type then we will get Error.

- ⦿ &, |, ^, ~, <<, >>

- ⦿ print(4&5) → Valid

- ⦿ print(10.5 & 5.6)
→ TypeError: unsupported operand type(s) for &: 'float' and 'float'

- ⦿ print(True & True) → Valid

- ⦿ & → If both bits are 1 then only result is 1 otherwise result is 0

- ⦿ | → If atleast one bit is 1 then result is 1 otherwise result is 0

- ⦿ ^ → If bits are different then only result is 1 otherwise result is 0

- ⦿ ~ → bitwise complement operator

- ⦿ 1 → 0 & 0 → 1

- ⦿ << → Bitwise Left Shift



⦿ >> → Bitwise Right Shift

- ⦿ `print(4&5) → 4`
- ⦿ `print(4|5) → 5`
- ⦿ `print(4^5) → 1`

Operator	Description
&	If both bits are 1 then only result is 1 otherwise result is 0
	If atleast one bit is 1 then result is 1 otherwise result is 0
^	If bits are different then only result is 1 otherwise result is 0
~	bitwise complement operator i.e 1 means 0 and 0 means 1
>>	Bitwise Left shift Operator
<<	Bitwise Right shift Operator

Bitwise Complement Operator (~):

We have to apply complement for total bits.

Eg: `print(~5) → -6`

Note:

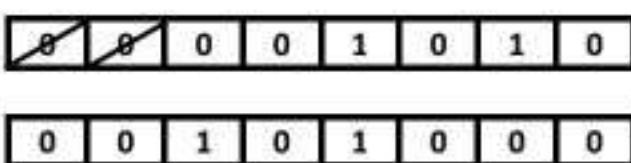
- ⦿ The most significant bit acts as sign bit. 0 value represents +ve number where as 1 represents -ve value.
- ⦿ Positive numbers will be represented directly in the memory where as -ve numbers will be represented indirectly in 2's complement form.

6) Shift Operators:

<< Left Shift Operator

After shifting the empty cells we have to fill with zero

`print(10<<2) → 40`

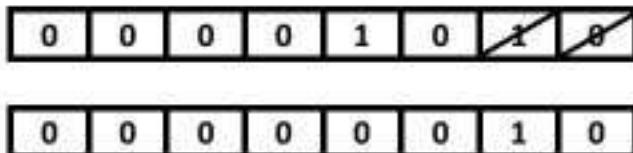




>> Right Shift Operator

After shifting the empty cells we have to fill with sign bit.(0 for +ve and 1 for -ve)

print(10>>2) → 2



We can apply bitwise operators for boolean types also

- ⦿ print(True & False) → False
- ⦿ print(True | False) → True
- ⦿ print(True ^ False) → True
- ⦿ print(~True) → -2
- ⦿ print(True<<2) → 4
- ⦿ print(True>>2) → 0

7) Assignment Operators:

- ⦿ We can use assignment operator to assign value to the variable.
Eg: x = 10

- ⦿ We can combine assignment operator with some other operator to form compound assignment operator.
Eg: x += 10 → x = x+10

The following is the list of all possible compound assignment operators in Python.

- +=
- -=
- *=
- /=
- %=
- //=
- **=
- &=
- |=
- ^=
- >>=
- <<=



Eg:

- 1) `x=10`
- 2) `x+=20`
- 3) `print(x) → 30`

Eg:

- 1) `x=10`
- 2) `x&=5`
- 3) `print(x) → 0`

8) Ternary Operator OR Conditional Operator

Syntax: `x = firstValue if condition else secondValue`

If condition is True then firstValue will be considered else secondValue will be considered.

Eg 1:

- 1) `a,b=10,20`
- 2) `x=30 if a<b else 40`
- 3) `print(x) #30`

Eg 2: Read two numbers from the keyboard and print minimum value -

- 1) `a=int(input("Enter First Number:"))`
- 2) `b=int(input("Enter Second Number:"))`
- 3) `min=a if a<b else b`
- 4) `print("Minimum Value:",min)`

Output:

Enter First Number:10
Enter Second Number:30
Minimum Value: 10

Note: Nesting of Ternary Operator is Possible.

Q) Program for Minimum of 3 Numbers

- 1) `a=int(input("Enter First Number:"))`
- 2) `b=int(input("Enter Second Number:"))`
- 3) `c=int(input("Enter Third Number:"))`



- 4) `min=a if a<b and a<c else b if b<c else c`
- 5) `print("Minimum Value:",min)`

Q) Program for Maximum of 3 Numbers

- 1) `a=int(input("Enter First Number:"))`
- 2) `b=int(input("Enter Second Number:"))`
- 3) `c=int(input("Enter Third Number:"))`
- 4) `max=a if a>b and a>c else b if b>c else c`
- 5) `print("Maximum Value:",max)`

Eg:

- 1) `a=int(input("Enter First Number:"))`
- 2) `b=int(input("Enter Second Number:"))`
- 3) `print("Both numbers are equal" if a==b else "First Number is Less than Second Number" if a<b else "First Number Greater than Second Number")`

Output:

D:\python_classes>py test.py

Enter First Number:10

Enter Second Number:10

Both numbers are equal

D:\python_classes>py test.py

Enter First Number:10

Enter Second Number:20

First Number is Less than Second Number

D:\python_classes>py test.py

Enter First Number:20

Enter Second Number:10

First Number Greater than Second Number

9) Special Operators:

Python defines the following 2 special operators

- 1) Identity Operators
- 2) Membership operators



1) Identity Operators

- We can use identity operators for address comparison.
- There are 2 identity operators are available
 - 1) is
 - 2) is not
- r1 is r2 returns True if both r1 and r2 are pointing to the same object.
- r1 is not r2 returns True if both r1 and r2 are not pointing to the same object.

Eg:

- 1) a=10
- 2) b=10
- 3) print(a is b) True
- 4) x=True
- 5) y=True
- 6) print(x is y) True

Eg:

- 1) a="durga"
- 2) b="durga"
- 3) print(id(a))
- 4) print(id(b))
- 5) print(a is b)

Eg:

- 1) list1=["one","two","three"]
- 2) list2=["one","two","three"]
- 3) print(id(list1))
- 4) print(id(list2))
- 5) print(list1 is list2) False
- 6) print(list1 is not list2) True
- 7) print(list1 == list2) True

Note: We can use is operator for address comparison where as == operator for content comparison.



2) Membership Operators:

- We can use Membership operators to check whether the given object present in the given collection. (It may be String, List, Set, Tuple OR Dict)
- In → Returns True if the given object present in the specified Collection
- not in → Returns True if the given object not present in the specified Collection

Eg:

- 1) `x="hello learning Python is very easy!!!"`
- 2) `print('h' in x)` True
- 3) `print('d' in x)` False
- 4) `print('d' not in x)` True
- 5) `print('Python' in x)` True

Eg:

- 1) `list1=["sunny","bunny","chinny","pinny"]`
- 2) `print("sunny" in list1)` True
- 3) `print("tunny" in list1)` False
- 4) `print("tunny" not in list1)` True

Operator Precedence:

If multiple operators present then which operator will be evaluated first is decided by operator precedence.

Eg:

`print(3+10*2)` → 23
`print((3+10)*2)` → 26

The following list describes operator precedence in Python

- 1) () → Parenthesis
- 2) ** → Exponential Operator
- 3) ~, - → Bitwise Complement Operator, Unary Minus Operator
- 4) *, /, %, // → Multiplication, Division, Modulo, Floor Division
- 5) +, - → Addition, Subtraction
- 6) <<, >> → Left and Right Shift
- 7) & → Bitwise And
- 8) ^ → Bitwise X-OR
- 9) | → Bitwise OR
- 10) >, >=, <, <=, ==, != → Relational OR Comparison Operators
- 11) =, +=, -=, *=... → Assignment Operators



- 12) is , is not → Identity Operators
- 13) in , not in → Membership operators
- 14) not → Logical not
- 15) and → Logical and
- 16) or → Logical or

- 1) a=30
- 2) b=20
- 3) c=10
- 4) d=5
- 5) `print((a+b)*c/d)` → 100.0
- 6) `print((a+b)*(c/d))` → 100.0
- 7) `print(a+(b*c)/d)` → 70.0
- 8)
- 9) `3/2*4+3+(10/5)**3-2`
- 10) `3/2*4+3+2.0**3-2`
- 11) `3/2*4+3+8.0-2`
- 12) `1.5*4+3+8.0-2`
- 13) `6.0+3+8.0-2`
- 14) 15.0

Mathematical Functions (math Module)

- ⦿ A Module is collection of functions, variables and classes etc.
- ⦿ math is a module that contains several functions to perform mathematical operations.
- ⦿ If we want to use any module in Python, first we have to import that module.
`import math`
- ⦿ Once we import a module then we can call any function of that module.

- 1) `import math`
- 2) `print(math.sqrt(16))`
- 3) `print(math.pi)`

Output

4.0
3.141592653589793

- ⦿ We can create alias name by using as keyword. `import math as m`
- ⦿ Once we create alias name, by using that we can access functions and variables of that module.

- 1) `import math as m`
- 2) `print(m.sqrt(16))`



3) `print(m.pi)`

- ⦿ We can import a particular member of a module explicitly as follows

```
from math import sqrt  
from math import sqrt,pi
```

- ⦿ If we import a member explicitly then it is not required to use module name while accessing.

- 1) `from math import sqrt,pi`
- 2) `print(sqrt(16))`
- 3) `print(pi)`
- 4) `print NameError: name 'math.pi' is not defined`

Important Functions of math Module:

- 1) `ceil(x)`
- 2) `floor(x)`
- 3) `pow(x,y)`
- 4) `factorial(x)`
- 5) `trunc(x)`
- 6) `gcd(x,y)`
- 7) `sin(x)`
- 8) `cos(x)`
- 9) `tan(x)`
- 10)

Important Variables of math Module:

`pi` → 3.14

`e` → 2.71

`inf` → infinity

`nan` → not a number

Q) Write a Python Program to find Area of Circle $\pi \cdot r^2$

- 1) `from math import pi`
- 2) `r = 16`
- 3) `print("Area of Circle is : ",pi*r**2)`

Output: Area of Circle is: 804.247719318987



Reading Dynamic Input from the Keyboard:

In Python 2 the following 2 functions are available to read dynamic input from the keyboard.

- 1) raw_input()
- 2) input()

1) raw_input():

This function always reads the data from the keyboard in the form of String Format. We have to convert that string type to our required type by using the corresponding type casting methods.

Eg: `x = raw_input("Enter First Number:")`
`print(type(x))` → It will always print str type only for any input type

2) input():

`input()` function can be used to read data directly in our required format. We are not required to perform type casting.

```
x = input("Enter Value")
type(x)
```

10 → int
"durga" → str
10.5 → float
True → bool

*****Note:**

- But in Python 3 we have only `input()` method and `raw_input()` method is not available.
- Python3 `input()` function behaviour exactly same as `raw_input()` method of Python2. i.e every input value is treated as str type only.
- `raw_input()` function of Python 2 is renamed as `input()` function in Python 3.

- ```
1) >>> type(input("Enter value:"))
2) Enter value:10
3) <class 'str'>
4)
5) Enter value:10.5
6) <class 'str'>
7)
8) Enter value:True
9) <class 'str'>
```

**Q) Write a program to read 2 numbers from the keyboard and print sum**

```
1) x=input("Enter First Number:")
2) y=input("Enter Second Number:")
3) i = int(x)
4) j = int(y)
5) print("The Sum:",i+j)
```

Enter First Number: 100

Enter Second Number: 200

The Sum: 300

---

```
1) x=int(input("Enter First Number:"))
2) y=int(input("Enter Second Number:"))
3) print("The Sum:",x+y)
```

```
print("The Sum:",int(input("Enter First Number:"))+int(input("Enter Second Number:")))
```

**Q) Write a Program to read Employee Data from the Keyboard and print that Data**

```
1) eno=int(input("Enter Employee No:"))
2) ename=input("Enter Employee Name:")
3) esal=float(input("Enter Employee Salary:"))
4) eaddr=input("Enter Employee Address:")
5) married=bool(input("Employee Married ?[True|False]:"))
6) print("Please Confirm Information")
7) print("Employee No :",eno)
8) print("Employee Name :",ename)
9) print("Employee Salary :",esal)
10) print("Employee Address :",eaddr)
11) print("Employee Married ? :",married)
```

D:\Python\_classes>py test.py

Enter Employee No:100

Enter Employee Name:Sunny

Enter Employee Salary:1000

Enter Employee Address:Mumbai

Employee Married ?[True|False]:True

Please Confirm Information



Employee No : 100  
Employee Name : Sunny  
Employee Salary : 1000.0  
Employee Address : Mumbai  
Employee Married ? : True

## How to read multiple values from the keyboard in a single line:

```
1) a,b= [int(x) for x in input("Enter 2 numbers :").split()]
2) print("Product is :", a*b)
```

```
D:\Python_classes>py test.py
Enter 2 numbers :10 20
Product is : 200
```

Note: split() function can take space as separator by default .But we can pass anything as separator.

## Q) Write a program to read 3 float numbers from the keyboard with, separator and print their sum

```
1) a,b,c= [float(x) for x in input("Enter 3 float numbers :").split(',')]
2) print("The Sum is :", a+b+c)
```

```
D:\Python_classes>py test.py
Enter 3 float numbers :10.5,20.6,20.1
The Sum is : 51.2
```

## eval():

eval Function take a String and evaluate the Result.

Eg: x = eval("10+20+30")
print(x)
Output: 60

Eg: x = eval(input("Enter Expression"))
Enter Expression: 10+2\*3/4
Output: 11.5
eval() can evaluate the Input to list, tuple, set, etc based the provided Input.



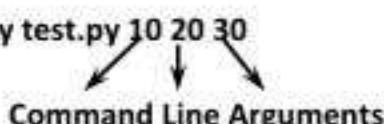
Eg: Write a Program to accept list from the keyboard on the display

- 1) `I = eval(input("Enter List"))`
- 2) `print(type(I))`
- 3) `print(I)`

## **COMMAND LINE ARGUMENTS**

- argv is not Array it is a List. It is available sys Module.
- The Argument which are passing at the time of execution are called Command Line Arguments.

Eg: D:\Python\_classes\py test.py 10 20 30



Within the Python Program this Command Line Arguments are available in argv. Which is present in SYS Module.

|         |    |    |    |
|---------|----|----|----|
| test.py | 10 | 20 | 30 |
|---------|----|----|----|

Note: argv[0] represents Name of Program. But not first Command Line Argument.  
argv[1] represent First Command Line Argument.

Program: To check type of argv from sys

```
import argv
print(type(argv))
```

D:\Python\_classes\py test.py

## **Write a Program to display Command Line Arguments**

- 1) `from sys import argv`
- 2) `print("The Number of Command Line Arguments:", len(argv))`
- 3) `print("The List of Command Line Arguments:", argv)`
- 4) `print("Command Line Arguments one by one:")`
- 5) `for x in argv:`
- 6) `print(x)`



```
D:\Python_classes>py test.py 10 20 30
The Number of Command Line Arguments: 4
The List of Command Line Arguments: ['test.py', '10','20','30']
Command Line Arguments one by one:
test.py
10
20
30
```

---

```
1) from sys import argv
2) sum=0
3) args=argv[1:]
4) for x in args :
5) n=int(x)
6) sum=sum+n
7) print("The Sum:",sum)
```

```
D:\Python_classes>py test.py 10 20 30 40
The Sum: 100
```

**Note 1:** Usually space is separator between command line arguments. If our command line argument itself contains space then we should enclose within double quotes (but not single quotes)

```
1) from sys import argv
2) print(argv[1])
```

```
D:\Python_classes>py test.py Sunny Leone
Sunny
```

```
D:\Python_classes>py test.py 'Sunny Leone'
'Sunny
```

```
D:\Python_classes>py test.py "Sunny Leone"
Sunny Leone
```

**Note 2:** Within the Python program command line arguments are available in the String form. Based on our requirement, we can convert into corresponding type by using type casting methods.

```
1) from sys import argv
2) print(argv[1]+argv[2])
3) print(int(argv[1])+int(argv[2]))
```



```
D:\Python_classes>py test.py 10 20
1020
30
```

**Note 3:** If we are trying to access command line arguments with out of range index then we will get Error.

- 1) `from sys import argv`
- 2) `print(argv[100])`

```
D:\Python_classes>py test.py 10 20
IndexError: list index out of range
```

**Note:** In Python there is argparse module to parse command line arguments and display some help messages whenever end user enters wrong input.

```
input()
raw_input()
```

#### Command Line Arguments

## Output Statements:

We can use `print()` function to display output.

**Form-1:** `print()` without any argument

Just it prints new line character

**Form-2:**

- 1) `print(String):`
- 2) `print("Hello World")`
- 3) We can use escape characters also
- 4) `print("Hello \n World")`
- 5) `print("Hello\tWorld")`
- 6) We can use repetition operator (\*) in the string
- 7) `print(10*"Hello")`
- 8) `print("Hello"*10)`
- 9) We can use + operator also
- 10) `print("Hello"+ "World")`

**Note:**

- ⦿ If both arguments are String type then + operator acts as concatenation operator.
- ⦿ If one argument is string type and second is any other type like int then we will get Error.
- ⦿ If both arguments are number type then + operator acts as arithmetic addition operator.

**Note:**

- 1) `print("Hello"+ "World")`
- 2) `print("Hello", "World")`

HelloWorld

Hello World

**Form-3: print() with variable number of arguments**

- 1) `a,b,c=10,20,30`
- 2) `print("The Values are :",a,b,c)`

**Output:** The Values are : 10 20 30

By default output values are separated by space. If we want we can specify separator by using "sep" attribute

- 1) `a,b,c=10,20,30`
- 2) `print(a,b,c,sep=',')`
- 3) `print(a,b,c,sep=':')`

D:\Python\_classes>py test.py

10,20,30

10:20:30

**Form-4: print() with end attribute**

- 1) `print("Hello")`
- 2) `print("Durga")`
- 3) `print("Soft")`

**Output:**

Hello

Durga

Soft



If we want output in the same line with space

- 1) `print("Hello",end=' ')`
- 2) `print("Durga",end=' ')`
- 3) `print("Soft")`

Output: Hello Durga Soft

Note: The default value for end attribute is \n, which is nothing but new line character.

#### Form-5: print(object) statement

We can pass any object (like list, tuple, set etc) as argument to the print() statement.

- 1) `l=[10,20,30,40]`
- 2) `t=(10,20,30,40)`
- 3) `print(l)`
- 4) `print(t)`

#### Form-6: print(String, variable list)

We can use print() statement with String and any number of arguments.

- 1) `s = "Durga"`
- 2) `a = 48`
- 3) `s1 ="Java"`
- 4) `s2 ="Python"`
- 5) `print("Hello",s,"Your Age is",a)`
- 6) `print("You are teaching",s1,"and",s2)`

#### Output:

Hello Durga Your Age is 48

You are teaching java and Python

#### Form-7: print (formatted string)

- 1) `%i → int`
- 2) `%d → int`
- 3) `%f → float`
- 4) `%s → String type`

Syntax: `print("formatted string" %(variable list))`



Eg 1:

```
1) a=10
2) b=20
3) c=30
4) print("a value is %i" %a)
5) print("b value is %d and c value is %d" %(b,c))
```

Output

a value is 10  
b value is 20 and c value is 30

Eg 2:

```
1) s="Durga"
2) list=[10,20,30,40]
3) print("Hello %s ...The List of Items are %s" %(s,list))
```

Output: Hello Durga ...The List of Items are [10, 20, 30, 40]

Form-8: print() with replacement operator {}

Eg:

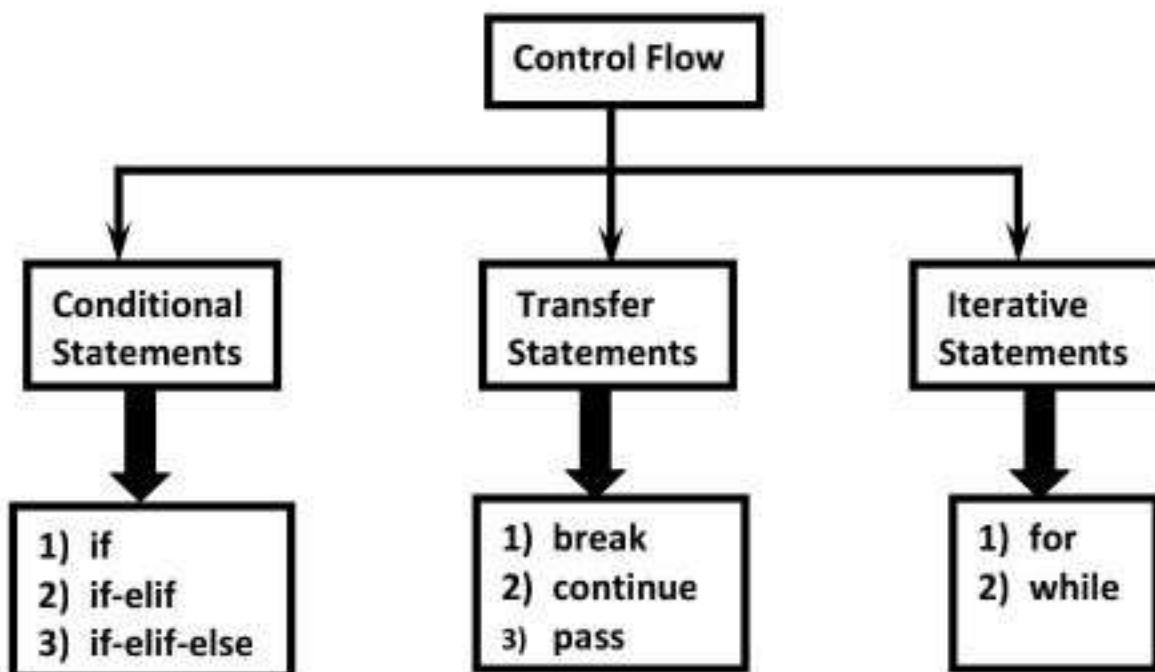
```
1) name = "Durga"
2) salary = 10000
3) gf = "Sunny"
4) print("Hello {0} your salary is {1} and Your Friend {2} is waiting".
 format(name,salary,gf))
5) print("Hello {x} your salary is {y} and Your Friend {z} is waiting".
 format(x=name,y=salary,z=gf))
```

Output

Hello Durga your salary is 10000 and Your Friend Sunny is waiting  
Hello Durga your salary is 10000 and Your Friend Sunny is waiting



Flow control describes the order in which statements will be executed at runtime.



## I. Conditional Statements

### 1) if

```
if condition : statement
OR
```

```
if condition :
 statement-1
 statement-2
 statement-3
```

If condition is true then statements will be executed.

Eg:

```
1) name=input("Enter Name:")
2) if name=="durga" :
3) print("Hello Durga Good Morning")
4) print("How are you!!!")
```

```
D:\Python_classes>py test.py
Enter Name:durga
Hello Durga Good Morning
How are you!!!
```



D:\Python\_classes>py test.py

Enter Name: Ravi

How are you!!!

## 2) if-else:

```
if condition:
 Action-1
else:
 Action-2
```

if condition is true then Action-1 will be executed otherwise Action-2 will be executed.

```
1) name=input("Enter Name:")
2) if name=="durga" :
3) print("Hello Durga Good Morning")
4) else:
5) print("Hello Guest Good Moring")
6) print("How are you!!!")
```

D:\Python\_classes>py test.py

Enter Name:durga

Hello Durga Good Morning

How are you!!!

D:\Python\_classes>py test.py

Enter Name:Ravi

Hello Guest Good Moring

How are you!!!

## 3) if-elif-else:

```
if condition1:
 Action-1
elif condition2:
 Action-2
elif condition3:
 Action-3
elif condition4:
 Action-4
...
else:
 Default Action
```

Based condition the corresponding action will be executed.



```
1) brand=input("Enter Your Favourite Brand:")
2) if brand=="RC":
3) print("It is childrens brand")
4) elif brand=="KF":
5) print("It is not that much kick")
6) elif brand=="FO":
7) print("Buy one get Free One")
8) else:
9) print("Other Brands are not recommended")
```

D:\Python\_classes>py test.py  
Enter Your Favourite Brand:RC  
It is childrens brand

D:\Python\_classes>py test.py  
Enter Your Favourite Brand:KF  
It is not that much kick

D:\Python\_classes>py test.py  
Enter Your Favourite Brand: KALYANI  
Other Brands are not recommended

**Note:**

- 1) else part is always optional. Hence the following are various possible syntaxes.
  - 1) If
  - 2) if – else
  - 3) if-elif-else
  - 4) if-elif
- 2) There is no switch statement in Python

**Q) Write a Program to find Biggest of given 2 Numbers from the Command Prompt?**

```
1) n1=int(input("Enter First Number:"))
2) n2=int(input("Enter Second Number:"))
3) if n1>n2:
4) print("Biggest Number is:",n1)
5) else :
6) print("Biggest Number is:",n2)
```

D:\Python\_classes>py test.py  
Enter First Number:10  
Enter Second Number:20  
Biggest Number is: 20



## Q) Write a Program to find Biggest of given 3 Numbers from the Command Prompt?

```
1) n1=int(input("Enter First Number:"))
2) n2=int(input("Enter Second Number:"))
3) n3=int(input("Enter Third Number:"))
4) if n1>n2 and n1>n3:
5) print("Biggest Number is:",n1)
6) elif n2>n3:
7) print("Biggest Number is:",n2)
8) else :
9) print("Biggest Number is:",n3)
```

D:\Python\_classes>py test.py

```
Enter First Number:10
Enter Second Number:20
Enter Third Number:30
Biggest Number is: 30
```

D:\Python\_classes>py test.py

```
Enter First Number:10
Enter Second Number:30
Enter Third Number:20
Biggest Number is: 30
```

Q) Write a program to find smallest of given 2 numbers?

Q) Write a program to find smallest of given 3 numbers?

Q) Write a program to check whether the given number is even or odd?

## Q) Write a Program to Check whether the given Number is in between 1 and 100?

```
1) n=int(input("Enter Number:"))
2) if n>=1 and n<=10 :
3) print("The number",n,"is in between 1 to 10")
4) else:
5) print("The number",n,"is not in between 1 to 10")
```



## Q) Write a Program to take a Single Digit Number from the Key Board and Print its Value in English Word?

```
1) 0 → ZERO
2) 1 → ONE
3)
4) n=int(input("Enter a digit from 0 to 9:"))
5) if n==0 :
6) print("ZERO")
7) elif n==1:
8) print("ONE")
9) elif n==2:
10) print("TWO")
11) elif n==3:
12) print("THREE")
13) elif n==4:
14) print("FOUR")
15) elif n==5:
16) print("FIVE")
17) elif n==6:
18) print("SIX")
19) elif n==7:
20) print("SEVEN")
21) elif n==8:
22) print("EIGHT")
23) elif n==9:
24) print("NINE")
25) else:
26) print("PLEASE ENTER A DIGIT FROM 0 TO 9")
```

## II. Iterative Statements

- ⦿ If we want to execute a group of statements multiple times then we should go for iterative statements.
- ⦿ Python supports 2 types of iterative statements.
  - 1) for loop
  - 2) while loop

### 1) for loop:

If we want to execute some action for every element present in some sequence (it may be string or collection) then we should go for for loop.

Syntax: for x in sequence:

    Body



Where sequence can be string or any collection.

Body will be executed for every element present in the sequence.

#### Eg 1: To print characters present in the given string

```
1) s="Sunny Leone"
2) for x in s:
3) print(x)
```

#### Output

S  
u  
n  
n  
y

L  
e  
o  
n  
e

#### Eg 2: To print characters present in string index wise:

```
1) s=input("Enter some String: ")
2) i=0
3) for x in s:
4) print("The character present at ",i,"index is:",x)
5) i=i+1
```

```
D:\Python_classes>py test.py
Enter some String: Sunny Leone
The character present at 0 index is : S
The character present at 1 index is : u
The character present at 2 index is : n
The character present at 3 index is : n
The character present at 4 index is : y
The character present at 5 index is :
The character present at 6 index is : L
The character present at 7 index is : e
The character present at 8 index is : o
The character present at 9 index is : n
The character present at 10 index is : e
```



**Eg 3:** To print Hello 10 times

```
1) for x in range(10):
2) print("Hello")
```

**Eg 4:** To display numbers from 0 to 10

```
1) for x in range(11):
2) print(x)
```

**Eg 5:** To display odd numbers from 0 to 20

```
1) for x in range(21):
2) if (x%2!=0):
3) print(x)
```

**Eg 6:** To display numbers from 10 to 1 in descending order

```
1) for x in range(10,0,-1):
2) print(x)
```

**Eg 7:** To print sum of numbers present inside list

```
1) list = eval(input("Enter List:"))
2) sum=0;
3) for x in list:
4) sum=sum+x;
5) print("The Sum=",sum)
```

```
D:\Python_classes>py test.py
Enter List:[10,20,30,40]
The Sum= 100
```

```
D:\Python_classes>py test.py
Enter List:[45,67]
The Sum= 112
```

## **2) while loop:**

If we want to execute a group of statements iteratively until some condition false, then we should go for while loop.

**Syntax:** while condition :  
 body



Eg: To print numbers from 1 to 10 by using while loop

```
1) x = 1
2) while x <= 10:
3) print(x)
4) x = x+1
```

Eg: To display the sum of first n numbers

```
1) n=int(input("Enter number:"))
2) sum=0
3) i=1
4) while i<=n:
5) sum=sum+i
6) i=i+1
7) print("The sum of first",n,"numbers is :",sum)
```

Eg: Write a program to prompt user to enter some name until entering Durga

```
1) name=""
2) while name!="durga":
3) name=input("Enter Name:")
4) print("Thanks for confirmation")
```

## Infinite Loops:

```
1) i=0;
2) while True :
3) i=i+1;
4) print("Hello",i)
```

## Nested Loops:

Sometimes we can take a loop inside another loop, which are also known as nested loops.

```
1) for i in range(4):
2) for j in range(4):
3) print("i=",i, " j=",j)
```

### Output

D:\Python\_classes>py test.py

```
i= 0 j= 0
i= 0 j= 1
i= 0 j= 2
```



```
i= 0 j= 3
i= 1 j= 0
i= 1 j= 1
i= 1 j= 2
i= 1 j= 3
i= 2 j= 0
i= 2 j= 1
i= 2 j= 2
i= 2 j= 3
i= 3 j= 0
i= 3 j= 1
i= 3 j= 2
i= 3 j= 3
```

### Q) Write a Program to display '\*'s in Right Angled Triangled Form

```
*
**


```

```
1) n = int(input("Enter number of rows:"))
2) for i in range(1,n+1):
3) for j in range(1,i+1):
4) print("*",end=" ")
5) print()
```

Alternative Way

```
1) n = int(input("Enter number of rows:"))
2) for i in range(1,n+1):
3) print("* " * i)
```

### Q) Write a Program to display '\*'s in Pyramid Style

(Also known as Equivalent Triangle)

```

*
**


```

```
1) n = int(input("Enter number of rows:"))
2) for i in range(1,n+1):
3) print(" " * (n-i),end="")
4) print("* " * i)
```



### III. Transfer Statements

#### 1) **break:**

We can use break statement inside loops to break loop execution based on some condition.

```
1) for i in range(10):
2) if i==7:
3) print("processing is enough..plz break")
4) break
5) print(i)
```

D:\Python\_classes>py test.py

```
0
1
2
3
4
5
6
processing is enough..plz break
```

Eg:

```
1) cart=[10,20,600,60,70]
2) for item in cart:
3) if item>500:
4) print("To place this order insurance must be required")
5) break
6) print(item)
```

D:\Python\_classes>py test.py

```
10
20
To place this order insurance must be required
```



## 2) continue:

We can use continue statement to skip current iteration and continue next iteration.

### Eg 1: To print odd numbers in the range 0 to 9

```
1) for i in range(10):
2) if i%2==0:
3) continue
4) print(i)
```

D:\Python\_classes>py test.py

1  
3  
5  
7  
9

### Eg 2:

```
1) cart=[10,20,500,700,50,60]
2) for item in cart:
3) if item>=500:
4) print("We cannot process this item :",item)
5) continue
6) print(item)
```

D:\Python\_classes>py test.py

10  
20  
We cannot process this item : 500  
We cannot process this item : 700  
50  
60

### Eg 3:

```
1) numbers=[10,20,0,5,0,30]
2) for n in numbers:
3) if n==0:
4) print("Hey how we can divide with zero..just skipping")
5) continue
6) print("100/{} = {}".format(n,100/n))
```



### Output

```
100/10 = 10.0
100/20 = 5.0
Hey how we can divide with zero..just skipping
100/5 = 20.0
Hey how we can divide with zero..just skipping
100/30 = 3.333333333333335
```

## Loops with else Block:

- Inside loop execution, if break statement not executed, then only else part will be executed.
- else means loop without break.

```
1) cart=[10,20,30,40,50]
2) for item in cart:
3) if item>=500:
4) print("We cannot process this order")
5) break
6) print(item)
7) else:
8) print("Congrats ...all items processed successfully")
```

### Output

```
10
20
30
40
50
Congrats ...all items processed successfully
```

### Eg:

```
1) cart=[10,20,600,30,40,50]
2) for item in cart:
3) if item>=500:
4) print("We cannot process this order")
5) break
6) print(item)
7) else:
8) print("Congrats ...all items processed successfully")
```



### Output

```
D:\Python_classes>py test.py
```

```
10
```

```
20
```

```
We cannot process this order
```

### **Q) What is the difference between for loop and while loop in Python?**

- ⦿ We can use loops to repeat code execution
- ⦿ Repeat code for every item in sequence → for loop
- ⦿ Repeat code as long as condition is true → while loop

**Q) How to exit from the loop? By using break statement**

**Q) How to skip some iterations inside loop? By using continue statement.**

**Q) When else part will be executed wrt loops? If loop executed without break**

### **3) pass statement:**

- pass is a keyword in Python.
- In our programming syntactically if block is required which won't do anything then we can define that empty block with pass keyword.

```
pass
|- It is an empty statement
|- It is null statement
|- It won't do anything
```

**Eg: if True:**

```
SyntaxError: unexpected EOF while parsing
if True: pass → valid
```

```
def m1():
```

```
SyntaxError: unexpected EOF while parsing
```

```
def m1(): pass
```



## Use Case of pass:

Sometimes in the parent class we have to declare a function with empty body and child class responsible to provide proper implementation. Such type of empty body we can define by using pass keyword. (It is something like abstract method in Java)

Eg: def m1(): pass

```
1) for i in range(100):
2) if i%9==0:
3) print(i)
4) else:pass
```

D:\Python\_classes>py test.py

```
0
9
18
27
36
45
54
63
72
81
90
99
```

## del Statement:

- del is a keyword in Python.
- After using a variable, it is highly recommended to delete that variable if it is no longer required, so that the corresponding object is eligible for Garbage Collection.
- We can delete variable by using del keyword.

```
1) x = 10
2) print(x)
3) del x
```

After deleting a variable we cannot access that variable otherwise we will get NameError.

```
1) x = 10
2) del x
3) print(x)
```

NameError: name 'x' is not defined.



**Note:** We can delete variables which are pointing to immutable objects. But we cannot delete the elements present inside immutable object.

- 1) `s = "durga"`
- 2) `print(s)`
- 3) `del s` → valid
- 4) `del s[0]` → `TypeError: 'str' object doesn't support item deletion`

## **Difference between del and None:**

In the case of `del`, the variable will be removed and we cannot access that variable(unbind operation)

- 1) `s = "durga"`
- 2) `del s`
- 3) `print(s)` → `NameError: name 's' is not defined.`

But in the case of `None` assignment the variable won't be removed but the corresponding object is eligible for Garbage Collection (re bind operation). Hence after assigning with `None` value, we can access that variable.

- 1) `s = "durga"`
- 2) `s = None`
- 3) `print(s)` → `None`



### Pattern-1: To print given number of \*s in a row

#### test.py

```
1) n=int(input('Enter n value:'))
2) for i in range(n):
3) print('*',end=' ')
```

#### Output:

Enter n value:5

\* \* \* \*

### Pattern-2: To print square pattern with \* symbols

#### test.py

```
1) n=int(input('Enter No Of Rows:'))
2) for i in range(n):
3) print('* '*n)
```

#### Output:

Enter No Of Rows:5

\* \* \* \*

\* \* \* \*

\* \* \* \*

\* \* \* \*

\* \* \* \*

### Pattern-3: To print square pattern with provided fixed digit in every row

#### test.py

```
1) n=int(input('Enter No Of Rows:'))
2) for i in range(n):
3) print((str(i+1)+' ')*n)
```



**Output:**

Enter No Of Rows:5

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

**Pattern-4: To print square pattern with alphabet symbols**

**test.py**

```
1) n=int(input('Enter No Of Rows:'))
2) for i in range(n):
3) print((chr(65+i)+ ' ')*n)
```

**Output:**

Enter No Of Rows:5

```
A A A A A
B B B B B
C C C C C
D D D D D
E E E E E
```

**Pattern-5: To print Right Angle Triangle pattern with \* symbols**

**test.py**

```
1) n=int(input('Enter No Of Rows:'))
2) for i in range(n):
3) for j in range(i+1):
4) print('*',end=' ')
5) print()
```

**Output:**

Enter No Of Rows:5

```
*
```

```
**
```

```

```



\*\*\*

### Pattern-6: To print Inverted Right Angle Triangle pattern with \* symbols

#### test.py

```
1) n=int(input('Enter No Of Rows:'))
2) for i in range(n):
3) print('*'*(n-i))
```

#### Output:

Enter No Of Rows:5

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

### Pattern-7: To print Pyramid pattern with \* symbols

#### test.py

```
1) n=int(input('Enter Number of rows:'))
2) for i in range(n):# 0,1,2,3
3) print(' '* (n-i-1))+ ('* ')*(i+1))
```

#### Output:

Enter number of rows:5

```

*
**


```



## Pattern-8: To print Inverted Pyramid Pattern with \* symbols

### test.py

```
1) n=int(input('Enter Number of Rows:'))
2) for i in range(n): #0,1,2,3
3) print('*'*i+'*'* (n-i))
```

### Output:

Enter Number of Rows:5

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

## Pattern-9: To print Diamond Pattern with \* symbols

### test.py

```
1) n=int(input('Enter n Value:'))
2) for i in range(n):#0,1,2,3
3) print('*'*(n-i-1)+'*'*(i+1))
4) for i in range(n-1):#0,1,2
5) print('*'*(i+1)+'*'*(n-i-1))
```

### Output:

Enter n Value:5

```

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



The most commonly used object in any project and in any programming language is String only. Hence we should aware complete information about String data type.

## What is String?

Any sequence of characters within either single quotes or double quotes is considered as a String.

### Syntax:

```
s = 'durga'
s = "durga"
```

**Note:** In most of other languages like C, C++, Java, a single character with in single quotes is treated as char data type value. But in Python we are not having char data type. Hence it is treated as String only.

### Eg:

```
>>> ch = 'a'
>>> type(ch)
<class 'str'>
```

## How to define multi-line String Literals?

We can define multi-line String literals by using triple single or double quotes.

### Eg:

```
>>> s = """durga
software
solutions""
```

We can also use triple quotes to use single quotes or double quotes as symbol inside String literal.

- 1) s = 'This is ' single quote symbol' → Invalid
- 2) s = 'This is \' single quote symbol' → Valid
- 3) s = "This is ' single quote symbol" → Valid
- 4) s = "This is " double quotes symbol" → Valid
- 5) s = 'The "Python Notes" by 'durga' is very helpful' → Invalid
- 6) s = "The "Python Notes" by 'durga' is very helpful" → Invalid
- 7) s = 'The \"Python Notes\" by \'durga\' is very helpful' → Valid
- 8) s = ""The "Python Notes" by 'durga' is very helpful"" → Valid



## How to Access Characters of a String?

We can access characters of a string by using the following ways.

- 1) By using index
- 2) By using slice operator

### 1) Accessing Characters By using Index:

- Python supports both +ve and -ve Index.
- +ve Index means Left to Right (Forward Direction)
- -ve Index means Right to Left (Backward Direction)

Eg: s = 'durga'

```
1) >>> s='durga'
2) >>> s[0]
3) 'd'
4) >>> s[4]
5) 'a'
6) >>> s[-1]
7) 'a'
8) >>> s[10]
9) IndexError: string index out of range
```

**Note:** If we are trying to access characters of a string with out of range index then we will get error saying: IndexError

### Q) Write a Program to Accept some String from the Keyboard and display its Characters by Index wise (both Positive and Negative Index)

test.py:

```
1) s=input("Enter Some String:")
2) i=0
3) for x in s:
4) print("The character present at positive index {} and at negative index {} is {}".format(i,i-len(s),x))
5) i=i+1
```

Output: D:\python\_classes>py test.py

Enter Some String:durga

The character present at positive index 0 and at negative index -5 is d

The character present at positive index 1 and at negative index -4 is u

The character present at positive index 2 and at negative index -3 is r

The character present at positive index 3 and at negative index -2 is g

The character present at positive index 4 and at negative index -1 is a



## 2) Accessing Characters by using Slice Operator:

- Syntax: s[bEginindex:endindex:step]
- Begin Index: From where we have to consider slice (substring)
- End Index: We have to terminate the slice (substring) at endindex-1
- Step: Incremented Value.

### Note:

- If we are not specifying bEgin index then it will consider from bEginning of the string.
- If we are not specifying end index then it will consider up to end of the string.
- The default value for step is 1.

```
1) >>> s="Learning Python is very very easy!!!"
2) >>> s[1:7:1]
3) 'earnin'
4) >>> s[1:7]
5) 'earnin'
6) >>> s[1:7:2]
7) 'eri'
8) >>> s[:7]
9) 'Learnin'
10)>>> s[7:]
11)'g Python is very very easy!!!'
12)>>> s[::-1]
13)'Learning Python is very very easy!!!'
14)>>> s[::-1]
15)'Learning Python is very very easy!!!'
16)>>> s[::-1]
17)'!!!ysae yrev yrev si nohtyP gniinraeL'
```

## Behaviour of Slice Operator:

- 1) s[bEgin:end:step]
- 2) Step value can be either +ve or -ve
- 3) If +ve then it should be forward direction(left to right) and we have to consider bEgin to end-1
- 4) If -ve then it should be backward direction (right to left) and we have to consider bEgin to end+1.

### \*\*\*Note:

- In the backward direction if end value is -1 then result is always empty.
- In the forward direction if end value is 0 then result is always empty.



## In Forward Direction:

default value for bEgin: 0

default value for end: length of string

default value for step: +1

## In Backward Direction:

default value for bEgin: -1

default value for end: -(length of string+1)

**Note:** Either forward or backward direction, we can take both +ve and -ve values for bEgin and end index.

## Slice Operator Case Study:

- 1) S = 'abcdefghijklm'
- 2) s[1:6:2] → 'bdf'
- 3) s[::-1] → 'lmkjihgfedcba'
- 4) s[::-1] → 'lmkjihgfedcba'
- 5) s[3:7:-1] → ''
- 6) s[7:4:-1] → 'hgf'
- 7) s[0:10000:1] → 'abcdefghijklm'
- 8) s[-4:1:-1] → 'gfedc'
- 9) s[-4:1:-2] → 'gec'
- 10) s[5:0:1] → ''
- 11) s[9:0:0] → ValueError: slice step cannot be zero
- 12) s[0:-10:-1] → ''
- 13) s[0:-11:-1] → 'a'
- 14) s[0:0:1] → ''
- 15) s[0:-9:-2] → ''
- 16) s[-5:-9:-2] → 'fd'
- 17) s[10:-1:-1] → ''
- 18) s[10000:2:-1] → 'lmkjihgfed'

**Note:** Slice operator never raises IndexError

## Mathematical Operators for String:

We can apply the following mathematical operators for Strings.

- 1) + operator for concatenation
- 2) \* operator for repetition

- print("durga"+"soft") → durgasoft
- print("durga"\*2) → durgadurga

**Note:**

- 1) To use + operator for Strings, compulsory both arguments should be str type.
- 2) To use \* operator for Strings, compulsory one argument should be str and other argument should be int.

## len() in-built Function:

We can use len() function to find the number of characters present in the string.

Eg:

```
s = 'durga'
print(len(s)) → 5
```

## Q) Write a Program to access each Character of String in Forward and Backward Direction by using while Loop?

```
1) s = "Learning Python is very easy !!!"
2) n = len(s)
3) i = 0
4) print("Forward direction")
5) while i < n:
6) print(s[i],end=' ')
7) i += 1
8) print("Backward direction")
9) i = -1
10) while i >= -n:
11) print(s[i],end=' ')
12) i = i-1
```

## Alternative ways:

```
1) s = "Learning Python is very easy !!!"
2) print("Forward direction")
3) for i in s:
4) print(i,end=' ')
5) print("Forward direction")
6) for i in s[::-1]:
7) print(i,end=' ')
8)
9) print("Backward direction")
10) for i in s[::-1]:
11) print(i,end=' ')
```



## Checking Membership:

We can check whether the character or string is the member of another string or not by using in and not in operators

```
s = 'durga'
print('d' in s) → True
print('z' in s) → False
```

```
1) s = input("Enter main string:")
2) subs = input("Enter sub string:")
3) if subs in s:
4) print(subs,"is found in main string")
5) else:
6) print(subs,"is not found in main string")
```

### Output:

```
D:\python_classes>py test.py
Enter main string:durgasoftwareolutions
Enter sub string:durga
durga is found in main string
```

```
D:\python_classes>py test.py
Enter main string:durgasoftwareolutions
Enter sub string:python
python is not found in main string
```

## Comparison of Strings:

- We can use comparison operators (<, <=, >, >=) and equality operators (==, !=) for strings.
- Comparison will be performed based on alphabetical order.

```
1) s1=input("Enter first string:")
2) s2=input("Enter Second string:")
3) if s1==s2:
4) print("Both strings are equal")
5) elif s1<s2:
6) print("First String is less than Second String")
7) else:
8) print("First String is greater than Second String")
```

### Output:

```
D:\python_classes>py test.py
Enter first string:durga
```



Enter Second string:durga

Both strings are equal

D:\python\_classes>py test.py

Enter first string:durga

Enter Second string:ravi

First String is less than Second String

D:\python\_classes>py test.py

Enter first string:durga

Enter Second string:anil

First String is greater than Second String

## Removing Spaces from the String:

We can use the following 3 methods

- 1) `rstrip()` → To remove spaces at right hand side
- 2) `lstrip()` → To remove spaces at left hand side
- 3) `strip()` → To remove spaces both sides

```
1) city=input("Enter your city Name:")
2) scity=city.strip()
3) if scity=='Hyderabad':
4) print("Hello Hyderbadi..Adab")
5) elif scity=='Chennai':
6) print("Hello Madrasi...Vanakkam")
7) elif scity=="Bangalore":
8) print("Hello Kannadiga...Shubhodaya")
9) else:
10) print("your entered city is invalid")
```

## Finding Substrings:

We can use the following 4 methods

### For forward direction:

- 1) `find()`
- 2) `index()`

### For backward direction:

- 1) `rfind()`
- 2) `rindex()`



## find():

s.find(substring)

Returns index of first occurrence of the given substring. If it is not available then we will get -1.

```
1) s="Learning Python is very easy"
2) print(s.find("Python")) #9
3) print(s.find("Java")) # -1
4) print(s.find("r"))#3
5) print(s.rfind("r"))#21
```

**Note:** By default find() method can search total string. We can also specify the boundaries to search.

s.find(substring,bEgin,end)

It will always search from bEgin index to end-1 index.

```
1) s="durgaravipavanshiva"
2) print(s.find('a'))#4
3) print(s.find('a',7,15))#10
4) print(s.find('z',7,15))#-1
```

## index():

index() method is exactly same as find() method except that if the specified substring is not available then we will get ValueError.

```
1) s=input("Enter main string:")
2) subs=input("Enter sub string:")
3) try:
4) n=s.index(subs)
5) except ValueError:
6) print("substring not found")
7) else:
8) print("substring found")
```

### Output:

```
D:\python_classes>py test.py
Enter main string:learning python is very easy
Enter sub string:python
substring found
```



```
D:\python_classes>py test.py
Enter main string:learning python is very easy
Enter sub string:java
substring not found
```

## Q) Program to display all Positions of Substring in a given Main String

```
1) s=input("Enter main string:")
2) subs=input("Enter sub string:")
3) flag=False
4) pos=-1
5) n=len(s)
6) while True:
7) pos=s.find(subs,pos+1,n)
8) if pos==-1:
9) break
10) print("Found at position",pos)
11) flag=True
12) if flag==False:
13) print("Not Found")
```

### Output:

```
D:\python_classes>py test.py
Enter main string:abbababababacdefg
Enter sub string:a
Found at position 0
Found at position 3
Found at position 5
Found at position 7
Found at position 9
Found at position 11
```

```
D:\python_classes>py test.py
Enter main string:abbababababacdefg
Enter sub string:bb
Found at position 1
```

## Counting substring in the given String:

We can find the number of occurrences of substring present in the given string by using count() method.

- 1) s.count(substring) → It will search through out the string.
- 2) s.count(substring, bEgin, end) → It will search from bEgin index to end-1 index.



```
1) s="abcabcbabcabcaadda"
2) print(s.count('a'))
3) print(s.count('ab'))
4) print(s.count('a',3,7))
```

**Output:**

6  
4  
2

**Replacing a String with another String:**

s.replace(oldstring, newstring)

inside s, every occurrence of old String will be replaced with new String.

**Eg 1:**

```
s = "Learning Python is very difficult"
s1 = s.replace("difficult","easy")
print(s1)
```

**Output:** Learning Python is very easy

**Eg 2: All occurrences will be replaced**

```
s = "ababababababab"
s1 = s.replace("a","b")
print(s1)
```

**Output:** bbbbbbbbbbbbbb

**Q) String Objects are Immutable then how we can change the Content by using replace() Method**

- Once we creates string object, we cannot change the content. This non changeable behaviour is nothing but immutability. If we are trying to change the content by using any method, then with those changes a new object will be created and changes won't be happened in existing object.
- Hence with replace() method also a new object got created but existing object won't be changed.

**Eg:**

```
s = "abab"
s1 = s.replace("a","b")
print(s,"is available at :",id(s))
print(s1,"is available at :",id(s1))
```



### Output:

```
abab is available at : 4568672
bbbb is available at : 4568704
```

In the above example, original object is available and we can see new object which was created because of replace() method.

### Splitting of Strings:

- We can split the given string according to specified separator by using split() method.
- `I = s.split(separator)`
- The default separator is space. The return type of split() method is List.

```
1) s="durga software solutions"
2) l=s.split()
3) for x in l:
4) print(x)
```

### Output:

```
durga
software
solutions
```

```
1) s="22-02-2018"
2) l=s.split('-')
3) for x in l:
4) print(x)
```

### Output:

```
22
02
2018
```

### Joining of Strings:

We can join a Group of Strings (List OR Tuple) wrt the given Separator.

```
s = separator.join(group of strings)
```

#### Eg 1:

```
t = ('sunny', 'bunny', 'chinny')
s = '-'.join(t)
print(s)
```

Output: sunny-bunny-chinny

**Eg 2:**

```
l = ['hyderabad', 'singapore', 'london', 'dubai']
s = ':'.join(l)
print(s)
```

**Output:** hyderabad:singapore:london:dubai

## **Changing Case of a String:**

We can change case of a string by using the following 4 methods.

- 1) `upper()` → To convert all characters to upper case
- 2) `lower()` → To convert all characters to lower case
- 3) `swapcase()` → Converts all lower case characters to upper case and all upper case characters to lower case
- 4) `title()` → To convert all character to title case. i.e first character in every word should be upper case and all remaining characters should be in lower case.
- 5) `capitalize()` → Only first character will be converted to upper case and all remaining characters can be converted to lower case

- ```
1) s = 'learning Python is very Easy'
2) print(s.upper())
3) print(s.lower())
4) print(s.swapcase())
5) print(s.title())
6) print(s.capitalize())
```

Output:

```
LEARNING PYTHON IS VERY EASY
```

Checking Starting and Ending Part of the String:

Python contains the following methods for this purpose

- 1) `s.startswith(substring)`
- 2) `s.endswith(substring)`

- ```
1) s = 'learning Python is very easy'
2) print(s.startswith('learning'))
3) print(s.endswith('learning'))
4) print(s.endswith('easy'))
```

**Output:**

True  
False  
True

## **To Check Type of Characters Present in a String:**

Python contains the following methods for this purpose.

- 1) **isalnum():** Returns True if all characters are alphanumeric( a to z , A to Z ,0 to 9 )
- 2) **isalpha():** Returns True if all characters are only alphabet symbols(a to z,A to Z)
- 3) **isdigit():** Returns True if all characters are digits only( 0 to 9 )
- 4) **islower():** Returns True if all characters are lower case alphabet symbols
- 5) **isupper():** Returns True if all characters are upper case aplhabet symbols
- 6) **istitle():** Returns True if string is in title case
- 7) **isspace():** Returns True if string contains only spaces

**Eg:**

- 1) print('Durga786'.isalnum()) → True
- 2) print('durga786'.isalpha()) → False
- 3) print('durga'.isalpha()) → True
- 4) print('durga'.isdigit()) → False
- 5) print('786786'.isdigit()) → True
- 6) print('abc'.islower()) → True
- 7) print('Abc'.islower()) → False
- 8) print('abc123'.islower()) → True
- 9) print('ABC'.isupper()) → True
- 10) print('Learning python is Easy'.istitle()) → False
- 11) print('Learning Python Is Easy'.istitle()) → True
- 12) print(' '.isspace()) → True

**Demo Program:**

```
1) s=input(" Enter any character:")
2) if s.isalnum():
3) print("Alpha Numeric Character")
4) if s.isalpha():
5) print("Alphabet character")
6) if s.islower():
7) print("Lower case alphabet character")
8) else:
9) print("Upper case alphabet character")
10) else:
11) print("it is a digit")
12) elif s.isspace():
```



```
13) print("It is space character")
14) else:
15) print("Non Space Special Character")
```

D:\python\_classes>py test.py

Enter any character:7  
Alpha Numeric Character  
it is a digit

D:\python\_classes>py test.py

Enter any character:a  
Alpha Numeric Character  
Alphabet character  
Lower case alphabet character

D:\python\_classes>py test.py

Enter any character:\$  
Non Space Special Character

D:\python\_classes>py test.py

Enter any character:A  
Alpha Numeric Character  
Alphabet character  
Upper case alphabet character

## Formatting the Strings:

We can format the strings with variable values by using replacement operator {} and format() method.

```
1) name = 'durga'
2) salary = 10000
3) age = 48
4) print("{} 's salary is {} and his age is {}".format(name,salary,age))
5) print("{} 's salary is {} and his age is {}".format(name,salary,age))
6) print("{} 's salary is {} and his age is {}".format(z=age,y=salary,x=name))
```

### Output:

durga 's salary is 10000 and his age is 48  
durga 's salary is 10000 and his age is 48  
durga 's salary is 10000 and his age is 48



## Important Programs regarding String Concept

### Q1) Write a Program to Reverse the given String

Input: durga

Output: agrud

#### 1<sup>st</sup> Way:

```
1) s = input("Enter Some String:")
2) print(s[::-1])
```

#### 2<sup>nd</sup> Way:

```
1) s = input("Enter Some String:")
2) print("."join(reversed(s)))
```

#### 3<sup>rd</sup> Way:

```
1) s = input("Enter Some String:")
2) i=len(s)-1
3) target=""
4) while i>=0:
5) target=target+s[i]
6) i=i-1
7) print(target)
```

### Q2) Program to Reverse Order of Words

Input: Learning Python is very Easy

Output: Easy Very is Python Learning

```
1) s=input("Enter Some String:")
2) l=s.split()
3) l1=[]
4) i=len(l)-1
5) while i>=0:
6) l1.append(l[i])
7) i=i-1
8) output=' '.join(l1)
9) print(output)
```



**Output:** Enter Some String:Learning Python is very easy!!  
easy!!! very is Python Learning

### **Q3) Program to Reverse Internal Content of each Word**

**Input:** Durga Software Solutions  
**Output:** agruD erawtfoS snoitulos

```
1) s=input("Enter Some String:")
2) l=s.split()
3) l1=[]
4) i=0
5) while i<len(l):
6) l1.append(l[i][::-1])
7) i=i+1
8) output=' '.join(l1)
9) print(output)
```

### **Q4) Write a Program to Print Characters at Odd Position and Even Position for the given String?**

#### **1<sup>st</sup> Way:**

```
s = input("Enter Some String:")
print("Characters at Even Position:",s[0::2])
print("Characters at Odd Position:",s[1::2])
```

#### **2<sup>nd</sup> Way:**

```
1) s=input("Enter Some String:")
2) i=0
3) print("Characters at Even Position:")
4) while i< len(s):
5) print(s[i],end=',')
6) i=i+2
7) print()
8) print("Characters at Odd Position:")
9) i=1
10) while i< len(s):
11) print(s[i],end=',')
12) i=i+2
```



## Q5) Program to Merge Characters of 2 Strings into a Single String by taking Characters alternatively

Input: s1 = "ravi"

s2 = "reja"

Output: rtaevjia

```
1) s1=input("Enter First String:")
2) s2=input("Enter Second String:")
3) output=""
4) i,j=0,0
5) while i<len(s1) or j<len(s2):
6) if i<len(s1):
7) output=output+s1[i]
8) i+=1
9) if j<len(s2):
10) output=output+s2[j]
11) j+=1
12) print(output)
```

### Output:

Enter First String:durga

Enter Second String:ravisoft

druarvgiasoft

## Q6) Write a Program to Sort the Characters of the String and First Alphabet Symbols followed by Numeric Values

Input: B4A1D3

Output: ABD134

```
1) s=input("Enter Some String:")
2) s1=s2=output=""
3) for x in s:
4) if x.isalpha():
5) s1=s1+x
6) else:
7) s2=s2+x
8) for x in sorted(s1):
9) output=output+x
10) for x in sorted(s2):
11) output=output+x
12) print(output)
```



### **Q7) Write a Program for the following Requirement**

Input: a4b3c2  
Output: aaaabbbcc

```
1) s=input("Enter Some String:")
2) output=""
3) for x in s:
4) if x.isalpha():
5) output=output+x
6) previous=x
7) else:
8) output=output+previous*(int(x)-1)
9) print(output)
```

**Note:** chr(unicode) → The corresponding character  
ord(character) → The corresponding unicode value

### **Q8) Write a Program to perform the following Activity**

Input: a4k3b2  
Output: aeknbd

```
1) s=input("Enter Some String:")
2) output=""
3) for x in s:
4) if x.isalpha():
5) output=output+x
6) previous=x
7) else:
8) output=output+chr(ord(previous)+int(x))
9) print(output)
```

### **Q9) Write a Program to Remove Duplicate Characters from the given Input String?**

Input: ABCDABBCDABBBCCCDDEEEF  
Output: ABCDEF

```
1) s = input("Enter Some String:")
2) l=[]
3) for x in s:
4) if x not in l:
5) l.append(x)
6) output="".join(l)
7) print(output)
```



## **Q10) Write a Program to find the Number of Occurrences of each Character present in the given String?**

Input: ABCABCABCDE

Output: A-3,B-4,C-3,D-1,E-1

```
1) s=input("Enter the Some String:")
2) d={}
3) for x in s:
4) if x in d.keys():
5) d[x]=d[x]+1
6) else:
7) d[x]=1
8) for k,v in d.items():
9) print("{} = {} Times".format(k,v))
```

## **Q11) Write a Program to perform the following Task?**

Input: 'one two three four five six seven'

Output: 'one owt three ruof five xis seven'

```
1) s = input('Enter Some String:')
2) l = s.split()
3) l1 = []
4) i = 0
5) while i<len(l):
6) if i%2==0:
7) l1.append(l[i])
8) else:
9) l1.append(l[i][::-1])
10) i=i+1
11) output=''.join(l1)
12) print('Original String:',s)
13) print('output String:',output)
```

### **Output:**

D:\durgaclasses>py test.py

Enter Some String:one two three four five six seven

Original String: one two three four five six seven

output String: one owt three ruof five xis seven



## Formatting the Strings:

- ⦿ We can format the strings with variable values by using replacement operator {} and format() method.
- ⦿ The main objective of format() method to format string into meaningful output form.

### Case- 1: Basic formatting for default, positional and keyword arguments

```
1) name = 'durga'
2) salary = 10000
3) age = 48
4) print("{} 's salary is {} and his age is {}".format(name,salary,age))
5) print("{0} 's salary is {1} and his age is {2}".format(name,salary,age))
6) print("{x} 's salary is {y} and his age is {z}".format(z=age,y=salary,x=name))
```

### Output:

durga 's salary is 10000 and his age is 48  
durga 's salary is 10000 and his age is 48  
durga 's salary is 10000 and his age is 48

### Case-2: Formatting Numbers

d → Decimal IntEger  
f → Fixed point number(float).The default precision is 6  
b → Binary format  
o → Octal Format  
x → Hexa Decimal Format (Lower case)  
X → Hexa Decimal Format (Upper case)

### Eg-1:

```
1) print("The intEger number is: {}".format(123))
2) print("The intEger number is: {:d}".format(123))
3) print("The intEger number is: {:5d}".format(123))
4) print("The intEger number is: {:05d}".format(123))
```

### Output:

The intEger number is: 123  
The intEger number is: 123  
The intEger number is: 123  
The intEger number is: 00123

**Eg-2:**

- 1) `print("The float number is: {}".format(123.4567))`
- 2) `print("The float number is: {:.f}".format(123.4567))`
- 3) `print("The float number is: {:.3f}".format(123.4567))`
- 4) `print("The float number is: {:08.3f}".format(123.4567))`
- 5) `print("The float number is: {:08.3f}".format(123.45))`
- 6) `print("The float number is: {:08.3f}".format(786786123.45))`

**Output:**

```
The float number is: 123.4567
The float number is: 123.456700
The float number is: 123.457
The float number is: 0123.457
The float number is: 0123.450
The float number is: 786786123.450
```

**Note:**

- ⦿ `{:08.3f}`
- ⦿ Total positions should be minimum 8.
- ⦿ After decimal point exactly 3 digits are allowed. If it is less than 8 then 0s will be placed in the last positions
- ⦿ If total number is < 8 positions then 0 will be placed in MSBs
- ⦿ If total number is >8 positions then all integral digits will be considered.
- ⦿ The extra digits we can take only 0

**Note:** For numbers default alignment is Right Alignment (>)

**Eg-3:** Print Decimal value in binary, octal and hexadecimal form

- 1) `print("Binary Form:{0:b}".format(153))`
- 2) `print("Octal Form:{0:o}".format(153))`
- 3) `print("Hexa decimal Form:{0:x}".format(154))`
- 4) `print("Hexa decimal Form:{0:X}".format(154))`

**Output:**

```
Binary Form:10011001
Octal Form:231
Hexa decimal Form:9a
Hexa decimal Form:9A
```

**Note:** We can represent only int values in binary, octal and hexadecimal and it is not possible for float values.

**Note:**

- 1) `{:5d}` It takes an intEger argument and assigns a minimum width of 5.
- 2) `{:8.3f}` It takes a float argument and assigns a minimum width of 8 including "." and after decimal point exactly 3 digits are allowed with round operation if required
- 3) `{:05d}` The blank places can be filled with 0. In this place only 0 allowed.

**Case-3: Number formatting for signed numbers**

- ⦿ While displaying positive numbers,if we want to include + then we have to write `{:+d}` and `{:+f}`
- ⦿ Using plus for -ve numbers there is no use and for -ve numbers - sign will come automatically.

- 1) `print("int value with sign:{:+d}".format(123))`
- 2) `print("int value with sign:{:+d}".format(-123))`
- 3) `print("float value with sign:{:+f}".format(123.456))`
- 4) `print("float value with sign:{:+f}".format(-123.456))`

**Output:**

```
int value with sign:+123
int value with sign:-123
float value with sign:+123.456000
float value with sign:-123.456000
```

**Case-4: Number formatting with alignment**

- ⦿ <, >, ^ and = are used for alignment
- ⦿ < → Left Alignment to the remaining space
- ⦿ ^ → Center alignment to the remaining space
- ⦿ > → Right alignment to the remaining space
- ⦿ = → Forces the signed(+) (-) to the left most position

**Note:** Default Alignment for numbers is Right Alignment.

**Ex:**

- 1) `print("{:5d}".format(12))`
- 2) `print("{:<5d}".format(12))`
- 3) `print("{:<05d}".format(12))`
- 4) `print("{:>5d}".format(12))`
- 5) `print("{:>05d}".format(12))`
- 6) `print("{:^5d}".format(12))`
- 7) `print("{:=5d}".format(-12))`
- 8) `print("{:^10.3f}".format(12.23456))`
- 9) `print("{:=-8.3f}".format(-12.23456))`



**Output:**

```
12
12
12000
12
00012
12
-12
12.235
- 12.235
```

**Case-5: String formatting with format()**

Similar to numbers, we can format String values also with format() method.

s.format(string)

```
1) print("{:5d}".format(12))
2) print("{:5}".format("rat"))
3) print("{:>5}".format("rat"))
4) print("{:<5}".format("rat"))
5) print("{:^5}".format("rat"))
6) print("{:.*5}".format("rat")) #Instead of * we can use any character(like +,$,a etc)
```

**Output:**

```
12
rat
rat
rat
rat
rat
```

**Note:** For numbers default alignment is right where as for strings default alignment is left

**Case-6: Truncating Strings with format() method**

```
1) print("{:.3}".format("durgasoftware"))
2) print("{:5.3}".format("durgasoftware"))
3) print("{:>5.3}".format("durgasoftware"))
4) print("{:^5.3}".format("durgasoftware"))
5) print("{:.*5.3}".format("durgasoftware"))
```

**Output:**

```
dur
dur
dur
```



dur  
\*dur\*

### Case-7: Formatting dictionary members using format()

```
1) person={'age':48,'name':'durga'}
2) print("{p[name]}'s age is: {p[age]}".format(p=person))
```

#### Output:

durga's age is: 48

Note: p is alias name of dictionary

person dictionary we are passing as keyword argument

More convenient way is to use \*\*person

```
1) person={'age':48,'name':'durga'}
2) print("{name}'s age is: {age}".format(**person))
```

Output: durga's age is: 48

### Case-8: Formatting class members using format()

```
1) class Person:
2) age=48
3) name="durga"
4) print("{p.name}'s age is :{p.age}".format(p=Person()))
```

Output: durga's age is :48

```
1) class Person:
2) def __init__(self,name,age):
3) self.name=name
4) self.age=age
5) print("{p.name}'s age is :{p.age}".format(p=Person('durga',48)))
6) print("{p.name}'s age is :{p.age}".format(p=Person('Ravi',50)))
```

Note: Here Person object is passed as keyword argument. We can access by using its reference variable in the template string

### Case-9: Dynamic Formatting using format()

```
1) string=":{fill}{align}{width}"
2) print(string.format('cat',fill='*',align='^',width=5))
3) print(string.format('cat',fill='*',align='^',width=6))
```



```
4) print(string.format('cat',fill='*',align='<',width=6))
5) print(string.format('cat',fill='*',align='>',width=6))
```

**Output:**

```
cat
*cat**
cat***
***cat
```

**Case-10: Dynamic Float format template**

```
1) num="{:align}{width}.{precision}f"
2) print(num.format(123.236,align='<',width=8,precision=2))
3) print(num.format(123.236,align='>',width=8,precision=2))
```

**Output:**

```
123.24
123.24
```

**Case-11: Formatting Date values**

```
1) import datetime
2) #datetime formatting
3) date=datetime.datetime.now()
4) print("It's now:{:%d/%m/%Y %H:%M:%S}".format(date))
```

**Output:** It's now:09/03/2018 12:36:26

**Case-12: Formatting complex numbers**

```
1) complexNumber=1+2j
2) print("Real Part:{0.real} and Imaginary Part:{0.imag}".format(complexNumber))
```

**Output:** Real Part: 1.0 and Imaginary Part: 2.0



### **Q1) Write a Program To REVERSE content of the given String by using slice operator?**

```
1) input: durga
2) output: agrud
3)
4) s = input('Enter Some String to Reverse:')
5) output = s[::-1]
6) print(output)
```

### **Q2) Write a Program To REVERSE content of the given String by using reversed() function?**

```
1) input: durga
2) output: agrud
3)
4) s=input('Enter Some String to Reverse:')
5) r=reversed(s)
6) output=".join(r)
7) print(output)
```

### **Q3) Write a Program To REVERSE content of the given String by using while loop?**

```
1) input: durga
2) output: agrud
3)
4) s=input('Enter Some String to Reverse:')
5) output=""
6) i=len(s)-1
7) while i>=0:
8) output=output+s[i]
9) i=i-1
10) print(output)
```



**Q4) Write a Program To REVERSE order of words present in the given string?**

```
1) input: Learning Python Is Very Easy
2) output: Easy Very Is Python Learning
3)
4) s=input('Enter Some String:')
5) l=s.split()
6) l1=l[::-1]
7) output=' '.join(l1)
8) print(output)
```

**Q5) Write a Program To REVERSE internal content of each word?**

```
1) input: 'Durga Software Solutions'
2) output: 'agruD erawtfoS snoituloS'
3)
4) s=input('Enter Any String:')
5) l=s.split()
6) l1=[]
7) for word in l:
8) l1.append(word[::-1])
9) output=' '.join(l1)
10) print(output)
```

**Q6) Write a Program To REVERSE internal content of every second word present in the given string?**

```
1) i/p: one two three four five six
2) o/p: one owt three ruof five xis
3)
4) s='one two three four five six'
5) l=s.split()
6) l1=[]
7) i=0
8) while i<len(l):
9) if i%2 == 0:
10) l1.append(l[i])
11) else:
```



```
12) l1.append(l[i][::-1])
13) i=i+1
14) output=''.join(l1)
15) print(output)
```

**Q7) Write a program to print the characters present at even index and odd index separately for the given string?**

**1<sup>st</sup> Way:**

```
1) s=input('Enter Input String:')
2) print('Characters present at Even Index:')
3) i=0
4) while i<len(s):
5) print(s[i])
6) i=i+2
7) print('Characters present at Odd Index:')
8) i=1
9) while i<len(s):
10) print(s[i])
11) i=i+2
```

**Output:**

```
D:\durgaclasses>py test.py
Enter Input String:durgasoftware
Characters present at Even Index:
d
r
a
o
t
a
e
Characters present at Odd Index:
u
g
s
f
w
r
```



**2<sup>nd</sup> Way:**

```
1) s=input('Enter Input String:')
2) print('Characters present at Even Index:',s[0::2])
3) print('Characters present at Even Index:',s[::2])
4) print('Characters present at Odd Index:',s[1::2])
```

**Q8) Write a program to merge characters of 2 strings into a single string by taking characters alternatively?**

Input:

```
s1='RAVI'
s2='TEJA'
```

Output: RTAEVJIA

**If strings are having same length:**

```
1) s1='RAVI'
2) s2='TEJA'
3) output=""
4) i,j=0,0
5) while i<len(s1) or j<len(s2):
6) output=output+s1[i]+s2[j]
7) i=i+1
8) j=j+1
9) print(output)
```

Output: RTAEVJIA

**2<sup>nd</sup> way by using map():**

```
1) s1='RAVI'
2) s2='TEJA'
3) l=list(map(lambda x,y:x+y,s1,s2))
4) print("".join(l))
```

**Note:** The above program can work if the lengths of 2 strings are same.



If strings having different lengths:

```
1) s1=input('Enter First String:')
2) s2=input('Enter Second String:')
3) output=""
4) i,j=0,0
5) while i<len(s1) or j<len(s2):
6) if i<len(s1):
7) output=output+s1[i]
8) i=i+1
9) if j<len(s2):
10) output=output+s2[j]
11) j=j+1
12) print(output)
```

Output:

```
D:\durgaclasses>py test.py
Enter First String:RAVIKIRAN
Enter Second String:TEJA
RTAEVJIAKIRAN
```

```
D:\durgaclasses>py test.py
Enter First String:RAVI
Enter Second String:TEJAKIRAN
RTAEVJIAKIRAN
```

Q9) Assume input string contains only alphabet symbols and digits.

Write a program to sort characters of the string, first alphabet symbols followed by digits?

```
1) input: B4A1D3
2) output: ABD134
3)
4) s='B4A1D3'
5) alphabets=[]
6) digits=[]
7) for ch in s:
8) if ch.isalpha():
9) alphabets.append(ch)
10) else:
```



```
11) digits.append(ch)
12) output=".join(sorted(alphabets)+sorted(digits))
13) print(output)
```

Alternative way:

```
1) s='B4A1D3'
2) alphabets=""
3) digits=""
4) for ch in s:
5) if ch.isalpha():
6) alphabets+=ch
7) else:
8) digits+=ch
9) output=""
10) for ch in sorted(alphabets):
11) output=output+ch
12) for ch in sorted(digits):
13) output=output+ch
14) print(output)
```

#### **Q10) Write a program for the following requirement?**

```
1) input: a4b3c2
2) output: aaaabbcc
3)
4) s=input('Enter Some String where alphabet symbol should be followed by digit:')
5) output=""
6) for ch in s:
7) if ch.isalpha():
8) x=ch
9) else:
10) d=int(ch)
11) output=output+x*d
12) print(output)
```



### **Q11) Write a program for the following requirement?**

```
1) input: a3z2b4
2) output: aaabbbbzz (sorted String)
3)
4) s=input('Enter Some String where alphabet symbol should be followed by digit:')
5) target=""
6) for ch in s:
7) if ch.isalpha():
8) x=ch
9) else:
10) d=int(ch)
11) target=target+x*d
12) output = ".join(sorted(target))"
13) print(output)
```

### **Q12) Write a program for the following requirement?**

```
1) input: aaaabbbccz
2) output: 4a3b2c1z
3)
4) s='aaaabbbccz'
5) output=''
6) previous=s[0]
7) c=1
8) i=1
9) while i<len(s):
10) if s[i]==previous:
11) c=c+1
12) else:
13) output=output+str(c)+previous
14) previous=s[i]
15) c=1
16) if i ==len(s)-1:
17) output=output+str(c)+previous
18) i=i+1
19) print(output)
```



### **Q13) Write a program for the following requirement?**

Input: a4k3b2

Output: aeknbd

In this example the following two functions are required to use

- 1) ord(): To find unicode value for the given character  
Eg: print(ord('a')) #97
- 2) chr(): To find corresponding character for the given unicode value  
Eg: print(chr(97)) # a

```
1) s='a4k3b2'
2) output=""
3) for ch in s:
4) if ch.isalpha():
5) x=ch
6) output=output+ch
7) else:
8) d=int(ch)
9) newc= chr(ord(x)+d)
10) output=output+newc
11) print(output)
```

### **Q14) Write a program to remove duplicate characters from the given input String?**

Input: AZZZBCDABBCDABBBBCCCCDDDDDEEEEEEF

Output: AZBCDEF

1<sup>st</sup> way:

```
1) s='AZZZBCDABBCDABBBBCCCCDDDDDEEEEEEF'
2) output=""
3) for ch in s:
4) if ch not in output:
5) output=output+ch
6) print(output) # AZBCDEF
```



2<sup>nd</sup> way:

```
1) s='AZZZBCDABBCDABBBBCCCCDDDDDEEEEEEF'
2) l=[]
3) for ch in s:
4) if ch not in l:
5) l.append(ch)
6) output=".join(l)
7) print(output) # AZBCDEF
```

3<sup>rd</sup> way by using set (but no guarantee for the order)

```
1) s='ABCDABXXXBCDABBBBCCCZZZCDDDDDEEEEEEF'
2) s1=set(s)
3) output=".join(s1)
4) print(output) #CAEZBFD
```

**Q15) Write a program to find the number of occurrences of each character present in the given string?**

By using count() method and List:

```
1) s='ABCDABXXXBCDABBBBCCCZZZCDDDDDEEEEEEF'
2) l=[]
3) for ch in s:
4) if ch not in l:
5) l.append(ch)
6)
7) for ch in sorted(l):
8) print('{} occurs {} times'.format(ch,s.count(ch)))
```

Without using count() method:

```
1) s='ABCDABXXXBCDABBBBCCCZZZCDDDDDEEEEEEF'
2) d={}
3) for ch in s:
4) d[ch]=d.get(ch,0)+1
5) for k,v in d.items():
6) print('{} occurs {} times'.format(k,v))
```



For sorting purpose:

```
1) for k,v in sorted(d.items()):
2) print('{} occurs {} times'.format(k,v))
```

**Q16) Write the program for the following requirement:**

Input: ABAABBBCA

Output: 4A3B1C

```
1) s='ABAABBBCA'
2) output=""
3) d={}
4) for ch in s:
5) d[ch]=d.get(ch,0)+1
6) for k,v in sorted(d.items()):
7) output=output+str(v)+k
8) print(output)
```

**Q17) Write the program for the following requirement:**

Input: ABAABBBCA

Output: A4B3C1

```
1) s='ABAABBBCA'
2) output=""
3) d={}
4) for ch in s:
5) d[ch]=d.get(ch,0)+1
6) for k,v in sorted(d.items()):
7) output=output+k+str(v)
8) print(output)
```

**Q18) Write a program to find the number of occurrences of each vowel present in the given string?**

```
1) s=input('Enter some string to search for vowels:')
2) v=['a','e','i','o','u','A','E','I','O','U']
3) d={}
4) for ch in s:
5) if ch in v:
```



```
6) d[ch]=d.get(ch,0)+1
7) for k,v in sorted(d.items()):
8) print('{} occurs {} times'.format(k,v))
```

```
D:\durgaclasses>py test.py
Enter some string to search for vowels:DURGASOFTWARESOLUTIONS
A occurs 2 times
E occurs 1 times
I occurs 1 times
O occurs 3 times
U occurs 2 times
```

```
D:\durgaclasses>py test.py
Enter some string to search for vowels:mississippi
i occurs 4 times
```

### **Q19) Write a program to check whether the given two strings are anagrams or not?**

Two strings are said to be anagrams iff both are having same content irrespective of characters position.

Eg: lazy and zaly

```
1) s1=input("Enter first string:")
2) s2=input("Enter second string:")
3) if(sorted(s1)==sorted(s2)):
4) print("The strings are anagrams.")
5) else:
6) print("The strings aren't anagrams.")
```

#### **Output:**

```
D:\durgaclasses>py test.py
Enter first string:lazy
Enter second string:zaly
The strings are anagrams.
```

```
D:\durgaclasses>py test.py
Enter first string:durga
Enter second string:urgadd
The strings aren't anagrams.
```



## Q20) Write a program to check whether the given string is palindrome or not ?

A string is said to be palindrome iff original string and its reversed strings are equal.

```
1) s=input("Enter Some string:")
2) if s==s[::-1]:
3) print('The given string is palindrome')
4) else:
5) print('The given string is not palindrome')
```

```
D:\durgaclasses>py test.py
Enter Some string:level
The given string is palindrome
```

```
D:\durgaclasses>py test.py
Enter Some string:madam
The given string is palindrome
```

```
D:\durgaclasses>py test.py
Enter Some string:apple
The given string is not palindrome
```

## Q21) Write the program for the following requirement:

```
1) inputs:
2) s1='abcdefg'
3) s2='xyz'
4) s3='12345'
5) output: ax1, by2,cz3,d4,e5,f,g
6)
7) s1='abcdefg'
8) s2='xyz'
9) s3='12345'
10) i=j=k=0
11) while i<len(s1) or j<len(s2) or k<len(s3):
12) output=""
13) if i<len(s1):
14) output=output+s1[i]
15) i=i+1
```



```
16) if j<len(s2):
17) output=output+s2[j]
18) j=j+1
19) if k<len(s3):
20) output=output+s3[k]
21) k=k+1
22) print(output)
```

Output:

ax1  
by2  
cz3  
d4  
e5  
f  
g



- ⦿ If we want to represent a group of individual objects as a single entity where insertion order preserved and duplicates are allowed, then we should go for List.
- ⦿ insertion order preserved.
- ⦿ duplicate objects are allowed.
- ⦿ heterogeneous objects are allowed.
- ⦿ List is dynamic because based on our requirement we can increase the size and decrease the size.
- ⦿ In List the elements will be placed within square brackets and with comma separator.
- ⦿ We can differentiate duplicate elements by using index and we can preserve insertion order by using index. Hence index will play very important role.
- ⦿ Python supports both positive and negative indexes. +ve index means from left to right where as negative index means right to left.

[10,"A","B",20, 30, 10]

|    |    |    |    |    |    |
|----|----|----|----|----|----|
| -6 | -5 | -4 | -3 | -2 | -1 |
| 10 | A  | B  | 20 | 30 | 10 |
| 0  | 1  | 2  | 3  | 4  | 5  |

- ⦿ List objects are mutable.i.e we can change the content.

## Creation of List Objects:

- 1) We can create empty list object as follows...

```
1) list=[]
2) print(list)
3) print(type(list))
4)
5) []
6) <class 'list'>
```

- 2) If we know elements already then we can create list as follows list = [10, 20, 30, 40]

- 3) With Dynamic Input:

```
1) list=eval(input("Enter List:"))
2) print(list)
3) print(type(list))
```

D:\Python\_classes>py test.py  
Enter List:[10,20,30,40]  
[10, 20, 30, 40]  
<class 'list'>



#### 4) With list() Function:

```
1) l=list(range(0,10,2))
2) print(l)
3) print(type(l))
```

```
D:\Python_classes>py test.py
[0, 2, 4, 6, 8]
<class 'list'>
```

Eg:

```
1) s="durga"
2) l=list(s)
3) print(l)
```

```
D:\Python_classes>py test.py
['d', 'u', 'r', 'g', 'a']
```

#### 5) With split() Function:

```
1) s="Learning Python is very very easy !!!"
2) l=s.split()
3) print(l)
4) print(type(l))
```

```
D:\Python_classes>py test.py
['Learning', 'Python', 'is', 'very', 'very', 'easy', '!!!']
<class 'list'>
```

**Note:** Sometimes we can take list inside another list, such type of lists are called nested lists.

```
[10, 20, [30, 40]]
```

## Accessing Elements of List:

We can access elements of the list either by using index or by using slice operator(:)

### 1) By using Index:

- ⦿ List follows zero based index. ie index of first element is zero.
- ⦿ List supports both +ve and -ve indexes.
- ⦿ +ve index meant for Left to Right
- ⦿ -ve index meant for Right to Left
- ⦿ list = [10, 20, 30, 40]



|        |    |    |    |    |
|--------|----|----|----|----|
| list → | 10 | 20 | 30 | 40 |
|        | 0  | 1  | 2  | 3  |

- ⌚ print(list[0]) → 10
- ⌚ print(list[-1]) → 40
- ⌚ print(list[10]) → IndexError: list index out of range

## 2) By using Slice Operator:

Syntax: list2 = list1[start:stop:step]

Start → It indicates the Index where slice has to Start  
Default Value is 0

Stop → It indicates the Index where slice has to End  
Default Value is max allowed Index of List ie Length of the List

Step → increment value  
Default Value is 1

- 1) n=[1,2,3,4,5,6,7,8,9,10]
- 2) print(n[2:7:2])
- 3) print(n[4::2])
- 4) print(n[3:7])
- 5) print(n[8:2:-2])
- 6) print(n[4:100])

### Output

```
D:\Python_classes>py test.py
[3, 5, 7]
[5, 7, 9]
[4, 5, 6, 7]
[9, 7, 5]
[5, 6, 7, 8, 9, 10]
```



## List vs Mutability:

Once we creates a List object, we can modify its content. Hence List objects are mutable.

- 1) n=[10,20,30,40]
- 2) print(n)
- 3) n[1]=777
- 4) print(n)

D:\Python\_classes>py test.py

[10, 20, 30, 40]

[10, 777, 30, 40]

## Traversing the Elements of List:

The sequential access of each element in the list is called traversal.

### **1) By using while Loop:**

- 1) n = [0,1,2,3,4,5,6,7,8,9,10]
- 2) i = 0
- 3) while i < len(n):
- 4) print(n[i])
- 5) i=i+1

D:\Python\_classes>py test.py

0  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10

### **2) By using for Loop:**

- 1) n=[0,1,2,3,4,5,6,7,8,9,10]
- 2) for n1 in n:
- 3) print(n1)



D:\Python\_classes>py test.py

```
0
1
2
3
4
5
6
7
8
9
10
```

### **3) To display only Even Numbers:**

```
1) n=[0,1,2,3,4,5,6,7,8,9,10]
2) for n1 in n:
3) if n1%2==0:
4) print(n1)
```

D:\Python\_classes>py test.py

```
0
2
4
6
8
10
```

### **4) To display Elements by Index wise:**

```
1) l = ["A","B","C"]
2) x = len(l)
3) for i in range(x):
4) print(l[i],"is available at positive index: ",i,"and at negative index: ",i-x)
```

D:\Python\_classes>py test.py

A is available at positive index: 0 and at negative index: -3  
B is available at positive index: 1 and at negative index: -2  
C is available at positive index: 2 and at negative index: -1



## Important Functions of List:

### I. To get Information about List:

#### 1) len():

Returns the number of elements present in the list

Eg: n = [10, 20, 30, 40]

print(len(n)) → 4

#### 2) count():

It returns the number of occurrences of specified item in the list

- 1) n=[1,2,2,2,2,3,3]
- 2) print(n.count(1))
- 3) print(n.count(2))
- 4) print(n.count(3))
- 5) print(n.count(4))

D:\Python\_classes>py test.py

1  
4  
2  
0

#### 3) index():

Returns the index of first occurrence of the specified item.

- 1) n = [1, 2, 2, 2, 2, 3, 3]
- 2) print(n.index(1)) → 0
- 3) print(n.index(2)) → 1
- 4) print(n.index(3)) → 5
- 5) print(n.index(4)) → ValueError: 4 is not in list

Note: If the specified element not present in the list then we will get ValueError. Hence before index() method we have to check whether item present in the list or not by using in operator.

print(4 in n) → False



## II. Manipulating Elements of List:

### 1) append() Function:

We can use append() function to add item at the end of the list.

```
1) list=[]
2) list.append("A")
3) list.append("B")
4) list.append("C")
5) print(list)
```

D:\Python\_classes>py test.py  
['A', 'B', 'C']

Eg: To add all elements to list upto 100 which are divisible by 10

```
1) list=[]
2) for i in range(101):
3) if i%10==0:
4) list.append(i)
5) print(list)
```

D:\Python\_classes>py test.py  
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

### 2) insert() Function:

To insert item at specified index position

```
1) n=[1,2,3,4,5]
2) n.insert(1,888)
3) print(n)
```

D:\Python\_classes>py test.py  
[1, 888, 2, 3, 4, 5]

```
1) n=[1,2,3,4,5]
2) n.insert(10,777)
3) n.insert(-10,999)
4) print(n)
```

D:\Python\_classes>py test.py  
[999, 1, 2, 3, 4, 5, 777]



**Note:** If the specified index is greater than max index then element will be inserted at last position. If the specified index is smaller than min index then element will be inserted at first position.

## Differences between append() and insert()

| append()                                                                           | insert()                                                     |
|------------------------------------------------------------------------------------|--------------------------------------------------------------|
| In List when we add any element it will come in last i.e. it will be last element. | In List we can insert any element in particular index number |

### 3) extend() Function:

To add all items of one list to another list

`l1.extend(l2)`

all items present in l2 will be added to l1

- 1) `order1=["Chicken","Mutton","Fish"]`
- 2) `order2=["RC","KF","FO"]`
- 3) `order1.extend(order2)`
- 4) `print(order1)`

```
D:\Python_classes>py test.py
['Chicken', 'Mutton', 'Fish', 'RC', 'KF', 'FO']
```

- 1) `order = ["Chicken","Mutton","Fish"]`
- 2) `order.extend("Mushroom")`
- 3) `print(order)`

```
D:\Python_classes>py test.py
['Chicken', 'Mutton', 'Fish', 'M', 'u', 's', 'h', 'r', 'o', 'o', 'm']
```

### 4) remove() Function:

We can use this function to remove specified item from the list. If the item present multiple times then only first occurrence will be removed.

- 1) `n=[10,20,10,30]`
- 2) `n.remove(10)`
- 3) `print(n)`

```
D:\Python_classes>py test.py
[20, 10, 30]
```

If the specified item not present in list then we will get `ValueError`



```
1) n=[10,20,10,30]
2) n.remove(40)
3) print(n)
```

ValueError: list.remove(x): x not in list

**Note:** Hence before using remove() method first we have to check specified element present in the list or not by using in operator.

## 5) pop() Function:

- It removes and returns the last element of the list.
- This is only function which manipulates list and returns some element.

```
1) n=[10,20,30,40]
2) print(n.pop())
3) print(n.pop())
4) print(n)
```

```
D:\Python_classes>py test.py
40
30
[10, 20]
```

If the list is empty then pop() function raises IndexError

```
1) n = []
2) print(n.pop()) → IndexError: pop from empty list
```

### Note:

- 1) pop() is the only function which manipulates the list and returns some value
- 2) In general we can use append() and pop() functions to implement stack datastructure by using list,which follows LIFO{Last In First Out} order.

In general we can use pop() function to remove last element of the list. But we can use to remove elements based on index.

n.pop(index) → To remove and return element present at specified index.  
n.pop() → To remove and return last element of the list

```
1) n = [10,20,30,40,50,60]
2) print(n.pop()) → 60
3) print(n.pop(1)) → 20
4) print(n.pop(10)) → IndexError: pop index out of range
```



## Differences between remove() and pop()

| remove()                                                     | pop()                                               |
|--------------------------------------------------------------|-----------------------------------------------------|
| 1) We can use to remove special element from the List.       | 1) We can use to remove last element from the List. |
| 2) It can't return any value.                                | 2) It returned removed element.                     |
| 3) If special element not available then we get VALUE ERROR. | 3) If List is empty then we get Error.              |

**Note:** List Objects are dynamic. i.e based on our requirement we can increase and decrease the size.

append(), insert(), extend() → for increasing the size/growable nature  
remove(), pop() → for decreasing the size /shrinking nature

## III) Ordering Elements of List:

### 1) reverse():

We can use to reverse() order of elements of list.

- 1) n=[10,20,30,40]
- 2) n.reverse()
- 3) print(n)

```
D:\Python_classes>py test.py
[40, 30, 20, 10]
```

### 2) sort():

In list by default insertion order is preserved. If want to sort the elements of list according to default natural sorting order then we should go for sort() method.

- For numbers → Default Natural sorting Order is Ascending Order
- For Strings → Default Natural sorting order is Alphabetical Order

- 1) n = [20,5,15,10,0]
- 2) n.sort()
- 3) print(n) → [0,5,10,15,20]
- 4)
- 5) s = ["Dog","Banana","Cat","Apple"]
- 6) s.sort()
- 7) print(s) → ['Apple','Banana','Cat','Dog']



**Note:** To use sort() function, compulsory list should contain only homogeneous elements. Otherwise we will get TypeError

- 1) `n=[20,10,"A","B"]`
- 2) `n.sort()`
- 3) `print(n)`

TypeError: '<' not supported between instances of 'str' and 'int'

**Note:** In Python 2 if List contains both numbers and Strings then sort() function first sort numbers followed by strings

- 1) `n=[20,"B",10,"A"]`
- 2) `n.sort()`
- 3) `print(n) # [10,20,'A','B']`

But in Python 3 it is invalid.

## To Sort in Reverse of Default Natural Sorting Order:

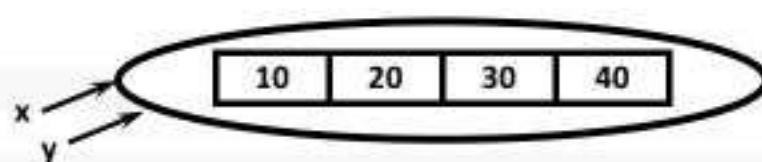
We can sort according to reverse of default natural sorting order by using `reverse=True` argument.

- 1) `n = [40,10,30,20]`
- 2) `n.sort()`
- 3) `print(n) → [10,20,30,40]`
- 4) `n.sort(reverse = True)`
- 5) `print(n) → [40,30,20,10]`
- 6) `n.sort(reverse = False)`
- 7) `print(n) → [10,20,30,40]`

## Aliasing and Cloning of List Objects:

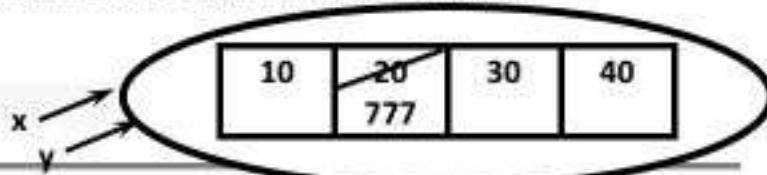
The process of giving another reference variable to the existing list is called aliasing.

- 1) `x=[10,20,30,40]`
- 2) `y=x`
- 3) `print(id(x))`
- 4) `print(id(y))`



The problem in this approach is by using one reference variable if we are changing content, then those changes will be reflected to the other reference variable.

- 1) `x = [10,20,30,40]`
- 2) `y = x`





- 3) `y[1] = 777`
- 4) `print(x) → [10,777,30,40]`

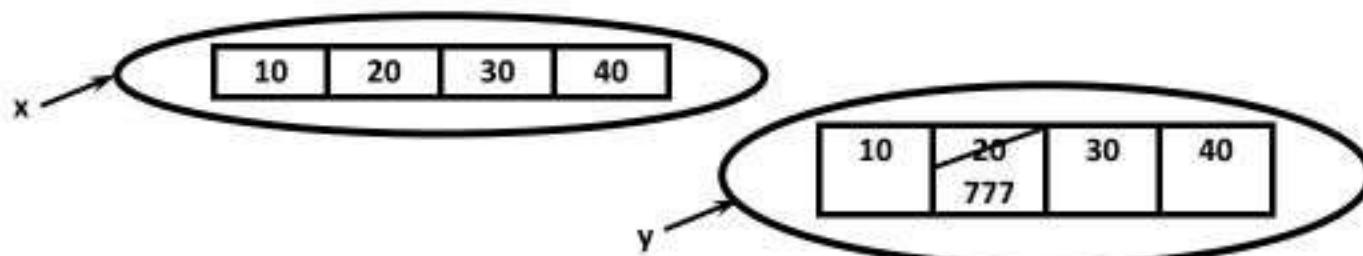
To overcome this problem we should go for cloning.

The process of creating exactly duplicate independent object is called cloning.

We can implement cloning by using slice operator or by using `copy()` function.

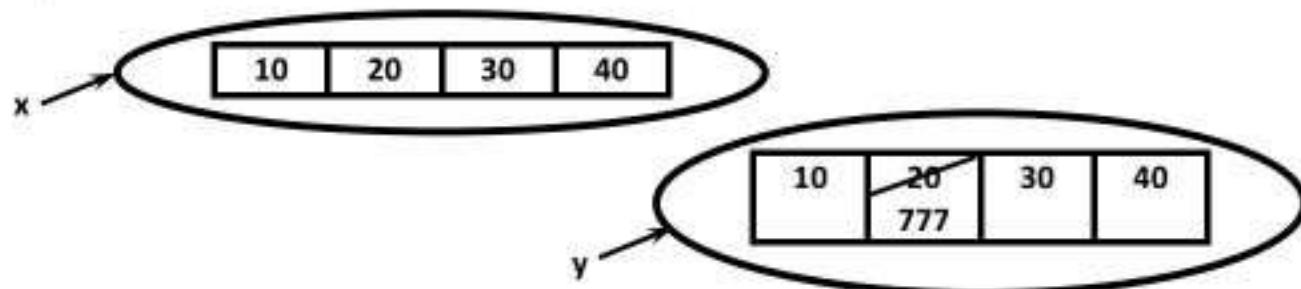
### 1) By using Slice Operator:

- 1) `x = [10,20,30,40]`
- 2) `y = x[:]`
- 3) `y[1] = 777`
- 4) `print(x) → [10, 20, 30, 40]`
- 5) `print(y) → [10, 777, 30, 40]`



### 2) By using `copy()` Function:

- 1) `x = [10,20,30,40]`
- 2) `y = x.copy()`
- 3) `y[1] = 777`
- 4) `print(x) → [10, 20, 30, 40]`
- 5) `print(y) → [10, 777, 30, 40]`



### Q) Difference between = Operator and `copy()` Function

- ⦿ = Operator meant for aliasing
- ⦿ `copy()` Function meant for cloning



## Using Mathematical Operators for List Objects:

We can use + and \* operators for List objects.

### 1) Concatenation Operator (+):

We can use + to concatenate 2 lists into a single list

- 1) `a = [10, 20, 30]`
- 2) `b = [40, 50, 60]`
- 3) `c = a+b`
- 4) `print(c) → [10, 20, 30, 40, 50, 60]`

**Note:** To use + operator compulsory both arguments should be list objects, otherwise we will get `TypeError`.

Eg:

`c = a+40 → TypeError: can only concatenate list (not "int") to list.`  
`c = a+[40] → Valid`

### 2) Repetition Operator (\*):

We can use repetition operator \* to repeat elements of list specified number of times.

- 1) `x = [10, 20, 30]`
- 2) `y = x*3`
- 3) `print(y) → [10, 20, 30, 10, 20, 30, 10, 20, 30]`

## Comparing List Objects

We can use comparison operators for List objects.

- 1) `x = ["Dog", "Cat", "Rat"]`
- 2) `y = ["Dog", "Cat", "Rat"]`
- 3) `z = ["DOG", "CAT", "RAT"]`
- 4) `print(x == y) → True`
- 5) `print(x == z) → False`
- 6) `print(x != z) → True`

**Note:** Whenever we are using comparison operators (==, !=) for List objects then the following should be considered

- 1) The Number of Elements
- 2) The Order of Elements
- 3) The Content of Elements (Case Sensitive)

**Note:** When ever we are using relational Operators (<, <=, >, >=) between List Objects, only 1<sup>ST</sup> Element comparison will be performed.



- 1) `x = [50, 20, 30]`
- 2) `y = [40, 50, 60, 100, 200]`
- 3) `print(x>y) → True`
- 4) `print(x>=y) → True`
- 5) `print(x<y) → False`
- 6) `print(x<=y) → False`

Eg:

- 1) `x = ["Dog", "Cat", "Rat"]`
- 2) `y = ["Rat", "Cat", "Dog"]`
- 3) `print(x>y) → False`
- 4) `print(x>=y) → False`
- 5) `print(x<y) → True`
- 6) `print(x<=y) → True`

## Membership Operators:

We can check whether element is a member of the list or not by using membership operators.

- 1) `in Operator`
- 2) `not in Operator`

- 1) `n=[10,20,30,40]`
- 2) `print (10 in n)`
- 3) `print (10 not in n)`
- 4) `print (50 in n)`
- 5) `print (50 not in n)`

### Output

True  
False  
False  
True

## clear() Function:

We can use clear() function to remove all elements of List.

- 1) `n=[10,20,30,40]`
- 2) `print(n)`
- 3) `n.clear()`
- 4) `print(n)`



### Output

```
D:\Python_classes>py test.py
[10, 20, 30, 40]
[]
```

## Nested Lists:

Sometimes we can take one list inside another list. Such type of lists are called nested lists.

```
1) n=[10,20,[30,40]]
2) print(n)
3) print(n[0])
4) print(n[2])
5) print(n[2][0])
6) print(n[2][1])
```

### Output

```
D:\Python_classes>py test.py
[10, 20, [30, 40]]
10
[30, 40]
30
40
```

**Note:** We can access nested list elements by using index just like accessing multi dimensional array elements.

## Nested List as Matrix:

In Python we can represent matrix by using nested lists.

```
1) n=[[10,20,30],[40,50,60],[70,80,90]]
2) print(n)
3) print("Elements by Row wise:")
4) for r in n:
5) print(r)
6) print("Elements by Matrix style:")
7) for i in range(len(n)):
8) for j in range(len(n[i])):
9) print(n[i][j],end=' ')
10) print()
```



### Output

```
D:\Python_classes>py test.py
[[10, 20, 30], [40, 50, 60], [70, 80, 90]]
```

Elements by Row wise:

```
[10, 20, 30]
[40, 50, 60]
[70, 80, 90]
```

Elements by Matrix style:

```
10 20 30
40 50 60
70 80 90
```

## List Comprehensions:

It is very easy and compact way of creating list objects from any iterable objects (Like List, Tuple, Dictionary, Range etc) based on some condition.

**Syntax:** list = [expression for item in list if condition]

```
1) s = [x*x for x in range(1,11)]
2) print(s)
3) v = [2**x for x in range(1,6)]
4) print(v)
5) m = [x for x in s if x%2==0]
6) print(m)
```

```
D:\Python_classes>py test.py
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
[2, 4, 8, 16, 32]
[4, 16, 36, 64, 100]
```

```
1) words=["Balaiah","Nag","Venkatesh","Chiranjeevi"]
2) l=[w[0] for w in words]
3) print(l)
```

**Output:** ['B', 'N', 'V', 'C']

```
1) num1=[10,20,30,40]
2) num2=[30,40,50,60]
3) num3=[i for i in num1 if i not in num2]
4) print(num3) [10,20]
5)
6) common elements present in num1 and num2
```



```
7) num4=[i for i in num1 if i in num2]
8) print(num4) [30, 40]
```

Eg:

```
1) words="the quick brown fox jumps over the lazy dog".split()
2) print(words)
3) l=[[w.upper(),len(w)] for w in words]
4) print(l)
```

#### Output

```
['the', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog']
[['THE', 3], ['QUICK', 5], ['BROWN', 5], ['FOX', 3], ['JUMPS', 5], ['OVER', 4],
 ['THE', 3], ['LAZY', 4], ['DOG', 3]]
```

#### Q) Write a Program to display Unique Vowels present in the given Word?

```
1) vowels=['a','e','i','o','u']
2) word=input("Enter the word to search for vowels: ")
3) found=[]
4) for letter in word:
5) if letter in vowels:
6) if letter not in found:
7) found.append(letter)
8) print(found)
9) print("The number of different vowels present in",word,"is",len(found))
```

```
D:\Python_classes>py test.py
Enter the word to search for vowels: durgasoftwareolutions
['u', 'a', 'o', 'e', 'i']
The number of different vowels present in durgasoftwareolutions is 5
```

List out all Functions of List and write a Program to use these Functions



- 1) Tuple is exactly same as List except that it is immutable. i.e once we creates Tuple object, we cannot perform any changes in that object.
- 2) Hence Tuple is Read only version of List.
- 3) If our data is fixed and never changes then we should go for Tuple.
- 4) Insertion Order is preserved
- 5) Duplicates are allowed
- 6) Heterogeneous objects are allowed.
- 7) We can preserve insertion order and we can differentiate duplicate objects by using index. Hence index will play very important role in Tuple also.
- 8) Tuple support both +ve and -ve index. +ve index means forward direction (from left to right) and -ve index means backward direction (from right to left)
- 9) We can represent Tuple elements within Parenthesis and with comma separator.
- 10) Parenthesis are optional but recommended to use.

```
1) t=10,20,30,40
2) print(t)
3) print(type(t))
4)
5) Output
6) (10, 20, 30, 40)
7)
8) <class 'tuple'>
9) t=()
10) print(type(t)) → tuple
```

**Note:** We have to take special care about single valued tuple.compulsory the value should ends with comma, otherwise it is not treated as tuple.

```
1) t=(10)
2) print(t)
3) print(type(t))
4)
5) Output
6) 10
7) <class 'int'>
```

**Eg:**

```
1) t=(10,)
2) print(t)
3) print(type(t))
4)
5) Output
6) (10,)
7) <class 'tuple'>
```



## **Q) Which of the following are valid Tuples?**

- 1) `t = ()`
- 2) `t = 10, 20, 30, 40`
- 3) `t = 10`
- 4) `t = 10,`
- 5) `t = (10)`
- 6) `t = (10,)`
- 7) `t = (10, 20, 30, 40)`

## **Tuple Creation:**

### **1) t = ()**

Creation of Empty Tuple

### **2) t = (10,)**

`t = 10,`

Creation of Single valued Tuple, Parenthesis are Optional, should ends with Comma

### **3) t = 10, 20, 30**

`t = (10, 20, 30)`

Creation of multi values Tuples & Parenthesis are Optional.

### **4) By using tuple() Function:**

- 1) `list=[10,20,30]`
- 2) `t=tuple(list)`
- 3) `print(t)`
- 4)
- 5) `t=tuple(range(10,20,2))`
- 6) `print(t)`

## **Accessing Elements of Tuple:**

We can access either by index or by slice operator

### **1) By using Index:**

- 1) `t = (10, 20, 30, 40, 50, 60)`
- 2) `print(t[0]) → 10`
- 3) `print(t[-1]) → 60`
- 4) `print(t[100]) → IndexError: tuple index out of range`



## 2) By using Slice Operator:

- 1) `t=(10,20,30,40,50,60)`
- 2) `print(t[2:5])`
- 3) `print(t[2:100])`
- 4) `print(t[::-2])`

### Output

(30, 40, 50)  
(30, 40, 50, 60)  
(10, 30, 50)

## Tuple vs Immutability:

- Once we creates tuple, we cannot change its content.
- Hence tuple objects are immutable.

### Eg:

`t = (10, 20, 30, 40)`  
`t[1] = 70` → `TypeError: 'tuple' object does not support item assignment`

## Mathematical Operators for Tuple:

We can apply + and \* operators for tuple

### 1) Concatenation Operator (+):

- 1) `t1=(10,20,30)`
- 2) `t2=(40,50,60)`
- 3) `t3=t1+t2`
- 4) `print(t3)` → `(10,20,30,40,50,60)`

### 2) Multiplication Operator OR Repetition Operator (\*)

- 1) `t1=(10,20,30)`
- 2) `t2=t1*3`
- 3) `print(t2)` → `(10,20,30,10,20,30,10,20,30)`



## Important Functions of Tuple:

### **1) len()**

To return number of elements present in the tuple.

Eg: `t = (10,20,30,40)`  
`print(len(t)) → 4`

### **2) count()**

To return number of occurrences of given element in the tuple

Eg: `t = (10, 20, 10, 10, 20)`  
`print(t.count(10)) → 3`

### **3) index()**

- Returns index of first occurrence of the given element.
- If the specified element is not available then we will get ValueError.

Eg: `t = (10, 20, 10, 10, 20)`  
`print(t.index(10)) → 0`  
`print(t.index(30)) → ValueError: tuple.index(x): x not in tuple`

### **4) sorted()**

To sort elements based on default natural sorting order

- 1) `t=(40,10,30,20)`
- 2) `t1=sorted(t)`
- 3) `print(t1)`
- 4) `print(t)`

#### Output

`[10, 20, 30, 40]`  
`(40, 10, 30, 20)`

We can sort according to reverse of default natural sorting order as follows

`t1 = sorted(t, reverse = True)`  
`print(t1) → [40, 30, 20, 10]`



## 5) min() And max() Functions:

These functions return min and max values according to default natural sorting order.

- 1) `t = (40,10,30,20)`
- 2) `print(min(t)) → 10`
- 3) `print(max(t)) → 40`

## 6) cmp():

- ⦿ It compares the elements of both tuples.
- ⦿ If both tuples are equal then returns 0
- ⦿ If the first tuple is less than second tuple then it returns -1
- ⦿ If the first tuple is greater than second tuple then it returns +1

- 1) `t1=(10,20,30)`
- 2) `t2=(40,50,60)`
- 3) `t3=(10,20,30)`
- 4) `print(cmp(t1,t2)) → -1`
- 5) `print(cmp(t1,t3)) → 0`
- 6) `print(cmp(t2,t3)) → +1`

Note: cmp() function is available only in Python2 but not in Python 3

## Tuple Packing and Unpacking:

We can create a tuple by packing a group of variables.

Eg:

```
a = 10
b = 20
c = 30
d = 40
t = a, b, c, d
print(t) → (10, 20, 30, 40)
```

- Here a, b, c, d are packed into a Tuple t. This is nothing but Tuple packing.
- Tuple unpacking is the reverse process of Tuple packing.
- We can unpack a Tuple and assign its values to different variables.

- 1) `t=(10,20,30,40)`
- 2) `a,b,c,d=t`
- 3) `print("a=",a,"b=",b,"c=",c,"d=",d)`

Output: a= 10 b= 20 c= 30 d= 40



**Note:** At the time of tuple unpacking the number of variables and number of values should be same, otherwise we will get ValueError.

**Eg:**

```
t = (10,20,30,40)
a, b, c = t → ValueError: too many values to unpack (expected 3)
```

## **Tuple Comprehension:**

- Tuple Comprehension is not supported by Python.
- `t = ( x**2 for x in range(1,6))`
- Here we are not getting tuple object and we are getting generator object.

```
1) t= (x**2 for x in range(1,6))
2) print(type(t))
3) for x in t:
4) print(x)
```

```
D:\Python_classes>py test.py
<class 'generator'>
1
4
9
16
25
```

## **Q) Write a Program to take a Tuple of Numbers from the Keyboard and Print its Sum and Average?**

```
1) t=eval(input("Enter Tuple of Numbers:"))
2) l=len(t)
3) sum=0
4) for x in t:
5) sum=sum+x
6) print("The Sum=",sum)
7) print("The Average=",sum/l)
```

```
D:\Python_classes>py test.py
Enter Tuple of Numbers:(10,20,30,40)
The Sum= 100
The Average= 25.0
```

```
D:\Python_classes>py test.py
Enter Tuple of Numbers: (100,200,300)
```



The Sum= 600

The Average= 200.0

## Differences between List and Tuple:

- List and Tuple are exactly same except small difference: List objects are mutable whereas Tuple objects are immutable.
- In both cases insertion order is preserved, duplicate objects are allowed, heterogeneous objects are allowed, index and slicing are supported.

| List                                                                                                                                            | Tuple                                                                                                                                                                                 |
|-------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1) List is a Group of Comma separated Values within Square Brackets and Square Brackets are mandatory.<br>Eg: <code>i = [10, 20, 30, 40]</code> | 1) Tuple is a Group of Comma separated Values within Parenthesis and Parenthesis are optional.<br>Eg: <code>t = (10, 20, 30, 40)</code><br><code>t = 10, 20, 30, 40</code>            |
| 2) List Objects are Mutable i.e. once we creates List Object we can perform any changes in that Object.<br>Eg: <code>i[1] = 70</code>           | 2) Tuple Objects are Immutable i.e. once we creates Tuple Object we cannot change its content.<br><code>t[1] = 70 → ValueError: tuple object does not support item assignment.</code> |
| 3) If the Content is not fixed and keep on changing then we should go for List.                                                                 | 3) If the content is fixed and never changes then we should go for Tuple.                                                                                                             |
| 4) List Objects can not used as Keys for Dictionaries because Keys should be Hashable and Immutable.                                            | 4) Tuple Objects can be used as Keys for Dictionaries because Keys should be Hashable and Immutable.                                                                                  |



- ❖ If we want to represent a group of unique values as a single entity then we should go for set.
- ❖ Duplicates are not allowed.
- ❖ Insertion order is not preserved. But we can sort the elements.
- ❖ Indexing and slicing not allowed for the set.
- ❖ Heterogeneous elements are allowed.
- ❖ Set objects are mutable i.e once we creates set object we can perform any changes in that object based on our requirement.
- ❖ We can represent set elements within curly braces and with comma separation
- ❖ We can apply mathematical operations like union, intersection, difference etc on set objects.

## Creation of Set Objects:

```
1) s={10,20,30,40}
2) print(s)
3) print(type(s))
```

### Output

```
{40, 10, 20, 30}
<class 'set'>
```

We can create set objects by using set() Function `s = set(any sequence)`

### Eg 1:

```
1) l = [10,20,30,40,10,20,10]
2) s=set(l)
3) print(s) # {40, 10, 20, 30}
```

### Eg 2:

```
1) s=set(range(5))
2) print(s) #{0, 1, 2, 3, 4}
```

### Note:

- ⦿ While creating empty set we have to take special care.
- ⦿ Compulsory we should use set() function.
- ⦿ `s = {}` → It is treated as dictionary but not empty set.

```
1) s={}
2) print(s)
3) print(type(s))
```



### Output

```
{}
<class 'dict'>
```

Eg:

```
1) s=set()
2) print(s)
3) print(type(s))
```

### Output

```
set()
<class 'set'>
```

## Important Functions of Set:

### 1) add(x):

Adds item x to the set.

```
1) s={10,20,30}
2) s.add(40);
3) print(s) #{40, 10, 20, 30}
```

### 2) update(x,y,z):

- To add multiple items to the set.
- Arguments are not individual elements and these are Iterable objects like List, Range etc.
- All elements present in the given Iterable objects will be added to the set.

```
1) s={10,20,30}
2) l=[40,50,60,10]
3) s.update(l,range(5))
4) print(s)
```

Output: {0, 1, 2, 3, 4, 40, 10, 50, 20, 60, 30}



## Q) What is the difference between add() and update() Functions in Set?

- We can use add() to add individual item to the Set, where as we can use update() function to add multiple items to Set.
- add() function can take only one argument where as update() function can take any number of arguments but all arguments should be iterable objects.

## Q) Which of the following are valid for set s?

- 1) s.add(10)
- 2) s.add(10,20,30) → TypeError: add() takes exactly one argument (3 given)
- 3) s.update(10) → TypeError: 'int' object is not iterable
- 4) s.update(range(1,10,2),range(0,10,2))

## 3) copy():

- Returns copy of the set.
- It is cloned object.

- 1) s = {10,20,30}
- 2) s1 = s.copy()
- 3) print(s1)

## 4) pop():

It removes and returns some random element from the set.

- 1) s={40,10,30,20}
- 2) print(s)
- 3) print(s.pop())
- 4) print(s)

### Output

```
{40, 10, 20, 30}
40
{10, 20, 30}
```

## 5) remove(x):

- It removes specified element from the set.
- If the specified element not present in the Set then we will get KeyError.

- 1) s = {40, 10, 30, 20}
- 2) s.remove(30)
- 3) print(s)



4) `s.remove(50)` `KeyError: 50`

## 6) `discard(x):`

- 1) It removes the specified element from the set.
- 2) If the specified element not present in the set then we won't get any error.

1) `s = {10, 20, 30}`  
2) `s.discard(10)`  
3) `print(s)` `20, 30`  
4) `s.discard(50)`  
5) `print(s)` `20, 30`

Q) What is the difference between `remove()` and `discard()` functions in Set?

Q) Explain differences between `pop()`, `remove()` and `discard()` functions in Set?

## 7) `clear():`

To remove all elements from the Set.

1) `s={10,20,30}`  
2) `print(s)`  
3) `s.clear()`  
4) `print(s)`

### Output

{10, 20, 30}  
set()

## Mathematical Operations on the Set:

### 1) `union():`

- `x.union(y)` → We can use this function to return all elements present in both sets
- `x.union(y)` OR `x|y`.

1) `x = {10, 20, 30, 40}`  
2) `y = {30, 40, 50, 60}`  
3) `print (x.union(y))` → {10, 20, 30, 40, 50, 60}  
4) `print (x|y)` → {10, 20, 30, 40, 50, 60}

### 2) `intersection():`

- `x.intersection(y)` OR `x&y`.
- Returns common elements present in both x and y.



- 1) `x = {10, 20, 30, 40}`
- 2) `y = {30, 40, 50, 60}`
- 3) `print(x.intersection(y)) → {40, 30}`
- 4) `print(x&y) → {40, 30}`

### 3) **difference():**

- `x.difference(y)` OR `x-y`.
- Returns the elements present in `x` but not in `y`.

- 1) `x = {10, 20, 30, 40}`
- 2) `y = {30, 40, 50, 60}`
- 3) `print(x.difference(y)) → 10, 20`
- 4) `print(x-y) → {10, 20}`
- 5) `print(y-x) → {50, 60}`

### 4) **symmetric\_difference():**

- `x.symmetric_difference(y)` OR `x^y`.
- Returns elements present in either `x` OR `y` but not in both.

- 1) `x = {10, 20, 30, 40}`
- 2) `y = {30, 40, 50, 60}`
- 3) `print(x.symmetric_difference(y)) → {10, 50, 20, 60}`
- 4) `print(x^y) → {10, 50, 20, 60}`

## **Membership Operators: (in, not in)**

- 1) `s=set("durga")`
- 2) `print(s)`
- 3) `print('d' in s)`
- 4) `print('z' in s)`

### **Output**

{'u', 'g', 'r', 'd', 'a'}

True

False

## **Set Comprehension:**

Set comprehension is possible.

- 1) `s = {x*x for x in range(5)}`
- 2) `print(s) → {0, 1, 4, 9, 16}`
- 3)



- 4) `s = {2**x for x in range(2,10,2)}`
- 5) `print(s) → {16, 256, 64, 4}`

## Set Objects won't support indexing and slicing:

- 1) `s = {10,20,30,40}`
- 2) `print(s[0]) → TypeError: 'set' object does not support indexing`
- 3) `print(s[1:3]) → TypeError: 'set' object is not subscriptable`

## Q) Write a Program to eliminate Duplicates Present in the List?

### Approach - 1

- 1) `l=eval(input("Enter List of values: "))`
- 2) `s=set(l)`
- 3) `print(s)`

```
D:\Python_classes>py test.py
Enter List of values: [10,20,30,10,20,40]
{40, 10, 20, 30}
```

### Approach - 2

- 1) `l=eval(input("Enter List of values: "))`
- 2) `l1=[]`
- 3) `for x in l:`
- 4)     `if x not in l1:`
- 5)         `l1.append(x)`
- 6) `print(l1)`

```
D:\Python_classes>py test.py
Enter List of values: [10,20,30,10,20,40]
[10, 20, 30, 40]
```

## Q) Write a Program to Print different Vowels Present in the given Word?

- 1) `w=input("Enter word to search for vowels: ")`
- 2) `s=set(w)`
- 3) `v={'a','e','i','o','u'}`
- 4) `d=s.intersection(v)`
- 5) `print("The different vowel present in",w,"are",d)`

```
D:\Python_classes>py test.py
Enter word to search for vowels: durga
The different vowel present in durga are {'u', 'a'}
```



- ⦿ We can use List, Tuple and Set to represent a group of individual objects as a single entity.
- ⦿ If we want to represent a group of objects as key-value pairs then we should go for Dictionary.

Eg:

- rollno ---- name
- phone number -- address
- ipaddress --- domain name

- ⦿ Duplicate keys are not allowed but values can be duplicated.
- ⦿ Hetrogeneous objects are allowed for both key and values.
- ⦿ Insertion order is not preserved
- ⦿ Dictionaries are mutable
- ⦿ Dictionaries are dynamic
- ⦿ indexing and slicing concepts are not applicable

**Note:** In C++ and Java Dictionaries are known as "Map" where as in Perl and Ruby it is known as "Hash"

## How to Create Dictionary?

- d = {} OR d = dict()
- We are creating empty dictionary. We can add entries as follows

```
1) d[100] = "durga"
2) d[200] = "ravi"
3) d[300] = "shiva"
4) print(d) → {100: 'durga', 200: 'ravi', 300: 'shiva'}
```

- If we know data in advance then we can create dictionary as follows
- d = {100:'durga', 200:'ravi', 300:'shiva'}
- d = {key:value, key:value}

## How to Access Data from the Dictionary?

We can access data by using keys.

```
1) d = {100: 'durga', 200: 'ravi', 300: 'shiva'}
2) print(d[100]) #durga
3) print(d[300]) #shiva
```

If the specified key is not available then we will get KeyError

```
print(d[400]) → KeyError: 400
```



We can prevent this by checking whether key is already available or not by using `has_key()` function or by using `in` operator.

`d.has_key(400)` → Returns 1 if key is available otherwise returns 0

But `has_key()` function is available only in Python 2 but not in Python 3. Hence compulsory we have to use `in` operator.

```
if 400 in d:
 print(d[400])
```

## Q) Write a Program to Enter Name and Percentage Marks in a Dictionary and Display Information on the Screen

```
1) rec={}
2) n=int(input("Enter number of students: "))
3) i=1
4) while i <=n:
5) name=input("Enter Student Name: ")
6) marks=input("Enter % of Marks of Student: ")
7) rec[name]=marks
8) i=i+1
9) print("Name of Student","\t","% of marks")
10) for x in rec:
11) print("\t",x,"\t\t",rec[x])
```

```
D:\Python_classes>py test.py
Enter number of students: 3
Enter Student Name: durga
Enter % of Marks of Student: 60%
Enter Student Name: ravi
Enter % of Marks of Student: 70%
Enter Student Name: shiva
Enter % of Marks of Student: 80%
```

| Name of Student | % of marks |
|-----------------|------------|
| durga           | 60%        |
| ravi            | 70 %       |
| shiva           | 80%        |



## How to Update Dictionaries?

- ⦿ `d[key] = value`
- ⦿ If the key is not available then a new entry will be added to the dictionary with the specified key-value pair
- ⦿ If the key is already available then old value will be replaced with new value.

```
1) d={100:"durga",200:"ravi",300:"shiva"}
2) print(d)
3) d[400]="pavan"
4) print(d)
5) d[100]="sunny"
6) print(d)
```

### Output

```
{100: 'durga', 200: 'ravi', 300: 'shiva'}
{100: 'durga', 200: 'ravi', 300: 'shiva', 400: 'pavan'}
{100: 'sunny', 200: 'ravi', 300: 'shiva', 400: 'pavan'}
```

## How to Delete Elements from Dictionary?

### 1) `del d[key]`

- It deletes entry associated with the specified key.
- If the key is not available then we will get `KeyError`.

```
1) d={100:"durga",200:"ravi",300:"shiva"}
2) print(d)
3) del d[100]
4) print(d)
5) del d[400]
```

### Output

```
{100: 'durga', 200: 'ravi', 300: 'shiva'}
{200: 'ravi', 300: 'shiva'}
KeyError: 400
```

### 2) `d.clear()`

To remove all entries from the dictionary.

```
1) d={100:"durga",200:"ravi",300:"shiva"}
2) print(d)
3) d.clear()
4) print(d)
```



### Output

```
{100: 'durga', 200: 'ravi', 300: 'shiva'}
{}
```

### **3) del d**

To delete total dictionary. Now we cannot access d.

```
1) d={100:"durga",200:"ravi",300:"shiva"}
2) print(d)
3) del d
4) print(d)
```

### Output

```
{100: 'durga', 200: 'ravi', 300: 'shiva'}
NameError: name 'd' is not defined
```

## **Important Functions of Dictionary:**

### **1) dict():**

To create a dictionary

- `d = dict()` → It creates empty dictionary
- `d = dict({100:"durga",200:"ravi"})` → It creates dictionary with specified elements
- `d = dict([(100,"durga"),(200,"shiva"),(300,"ravi")])`  
→ It creates dictionary with the given list of tuple elements

### **2) len()**

Returns the number of items in the dictionary.

### **3) clear():**

To remove all elements from the dictionary.

### **4) get():**

To get the value associated with the key

**d.get(key)**

If the key is available then returns the corresponding value otherwise returns None. It wont raise any error.



### d.get(key,defaultvalue)

If the key is available then returns the corresponding value otherwise returns default value.

- 1) `d={100:"durga",200:"ravi",300:"shiva"}`
- 2) `print(d[100])` → durga
- 3) `print(d[400])` → KeyError:400
- 4) `print(d.get(100))` → durga
- 5) `print(d.get(400))` → None
- 6) `print(d.get(100,"Guest"))` → durga
- 7) `print(d.get(400,"Guest"))` → Guest

### 5) pop():

`d.pop(key)`

- It removes the entry associated with the specified key and returns the corresponding value.
- If the specified key is not available then we will get KeyError.

- 1) `d={100:"durga",200:"ravi",300:"shiva"}`
- 2) `print(d.pop(100))`
- 3) `print(d)`
- 4) `print(d.pop(400))`

#### Output

```
durga
{200: 'ravi', 300: 'shiva'}
KeyError: 400
```

### 6) popitem():

It removes an arbitrary item(key-value) from the dictionary and returns it.

- 1) `d={100:"durga",200:"ravi",300:"shiva"}`
- 2) `print(d)`
- 3) `print(d.popitem())`
- 4) `print(d)`

#### Output

```
{100: 'durga', 200: 'ravi', 300: 'shiva'}
(300, 'shiva')
{100: 'durga', 200: 'ravi'}
```

If the dictionary is empty then we will get KeyError

```
d={}
print(d.popitem()) ==>KeyError: 'popitem(): dictionary is empty'
```



## **7) keys():**

It returns all keys associated with dictionary.

- 1) `d={100:"durga",200:"ravi",300:"shiva"}`
- 2) `print(d.keys())`
- 3) `for k in d.keys():`
- 4) `print(k)`

### **Output**

```
dict_keys([100, 200, 300])
100
200
300
```

## **8) values():**

It returns all values associated with the dictionary.

- 1) `d={100:"durga",200:"ravi",300:"shiva"}`
- 2) `print(d.values())`
- 3) `for v in d.values():`
- 4) `print(v)`

### **Output**

```
dict_values(['durga', 'ravi', 'shiva'])
durga
ravi
shiva
```

## **9) items():**

It returns list of tuples representing key-value pairs.

`[(k,v),(k,v),(k,v)]`

- 1) `d={100:"durga",200:"ravi",300:"shiva"}`
- 2) `for k,v in d.items():`
- 3) `print(k,"--",v)`

### **Output**

```
100 -- durga
200 -- ravi
300 -- shiva
```



## 10) copy():

To create exactly duplicate dictionary (cloned copy)

`d1 = d.copy();`

## 11) setdefault():

`d.setdefault(k,v)`

- If the key is already available then this function returns the corresponding value.
- If the key is not available then the specified key-value will be added as new item to the dictionary.

```
1) d={100:"durga",200:"ravi",300:"shiva"}
2) print(d.setdefault(400,"pavan"))
3) print(d)
4) print(d.setdefault(100,"sachin"))
5) print(d)
```

### Output

```
pavan
{100: 'durga', 200: 'ravi', 300: 'shiva', 400: 'pavan'}
durga
{100: 'durga', 200: 'ravi', 300: 'shiva', 400: 'pavan'}
```

## 12) update():

`d.update(x)`

All items present in the dictionary x will be added to dictionary d

## Q) Write a Program to take Dictionary from the Keyboard and print the Sum of Values?

```
1) d=eval(input("Enter dictionary:"))
2) s=sum(d.values())
3) print("Sum= ",s)
```

### Output

```
D:\Python_classes>py test.py
Enter dictionary:{'A':100,'B':200,'C':300}
Sum= 600
```



**Q) Write a Program to find Number of Occurrences of each Letter present in the given String?**

```
1) word=input("Enter any word: ")
2) d={}
3) for x in word:
4) d[x]=d.get(x,0)+1
5) for k,v in d.items():
6) print(k,"occurred ",v," times")
```

**Output**

```
D:\Python_classes>py test.py
Enter any word: mississippi
m occurred 1 times
i occurred 4 times
s occurred 4 times
p occurred 2 times
```

**Q) Write a Program to find Number of Occurrences of each Vowel present in the given String?**

```
1) word=input("Enter any word: ")
2) vowels={'a','e','i','o','u'}
3) d={}
4) for x in word:
5) if x in vowels:
6) d[x]=d.get(x,0)+1
7) for k,v in sorted(d.items()):
8) print(k,"occurred ",v," times")
```

**Output**

```
D:\Python_classes>py test.py
Enter any word: doganimaldoganimal
a occurred 4 times
i occurred 2 times
o occurred 2 times
```



**Q) Write a Program to accept Student Name and Marks from the Keyboard and creates a Dictionary. Also display Student Marks by taking Student Name as Input?**

```
1) n=int(input("Enter the number of students: "))
2) d={}
3) for i in range(n):
4) name=input("Enter Student Name: ")
5) marks=input("Enter Student Marks: ")
6) d[name]=marks
7) while True:
8) name=input("Enter Student Name to get Marks: ")
9) marks=d.get(name,-1)
10) if marks== -1:
11) print("Student Not Found")
12) else:
13) print("The Marks of",name,"are",marks)
14) option=input("Do you want to find another student marks[Yes|No]")
15) if option=="No":
16) break
17) print("Thanks for using our application")
```

#### **Output**

D:\Python\_classes>py test.py  
Enter the number of students: 5

Enter Student Name: sunny  
Enter Student Marks: 90

Enter Student Name: banny  
Enter Student Marks: 80

Enter Student Name: chinny  
Enter Student Marks: 70

Enter Student Name: pinny  
Enter Student Marks: 60

Enter Student Name: vinny  
Enter Student Marks: 50

Enter Student Name to get Marks: sunny  
The Marks of sunny are 90



Do you want to find another student marks[Yes | No]Yes

Enter Student Name to get Marks: durga

Student Not Found

Do you want to find another student marks[Yes | No]No

Thanks for using our application

## Dictionary Comprehension:

Comprehension concept applicable for dictionaries also.

- 1) squares={x:x\*x for x in range(1,6)}
- 2) print(squares)
- 3) doubles={x:2\*x for x in range(1,6)}
- 4) print(doubles)

### Output

{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

{1: 2, 2: 4, 3: 6, 4: 8, 5: 10}



- ⦿ If a group of statements is repeatedly required then it is not recommended to write these statements everytime separately. We have to define these statements as a single unit and we can call that unit any number of times based on our requirement without rewriting. This unit is nothing but function.
- ⦿ The main advantage of functions is code Reusability.
- ⦿ Note: In other languages functions are known as methods, procedures, subroutines etc
- ⦿ Python supports 2 types of functions
  - 1) Built in Functions
  - 2) User Defined Functions

## 1) Built in Functions:

The functions which are coming along with Python software automatically, are called built in functions or pre defined functions.

Eg: id()

type()  
input()  
eval()  
etc..

## 2) User Defined Functions:

The functions which are developed by programmer explicitly according to business requirements, are called user defined functions.

### Syntax to Create User defined Functions:

```
def function_name(parameters):
 """ doc string """
 ...
 ...
 return value
```

Note: While creating functions we can use 2 keywords

- 1) def (mandatory)
- 2) return (optional)

Eg 1: Write a function to print Hello

**test.py**

- ```
1) def wish():
2)     print("Hello Good Morning")
```



- 3) wish()
- 4) wish()
- 5) wish()

Parameters

Parameters are inputs to the function. If a function contains parameters, then at the time of calling, compulsory we should provide values otherwise, otherwise we will get error.

Eg: Write a function to take name of the student as input and print wish message by name.

```
1) def wish(name):  
2)     print("Hello",name," Good Morning")  
3) wish("Durga")  
4) wish("Ravi")
```

```
D:\Python_classes>py test.py  
Hello Durga Good Morning  
Hello Ravi Good Morning
```

Eg: Write a function to take number as input and print its square value

```
1) def squareIt(number):  
2)     print("The Square of",number,"is", number*number)  
3) squareIt(4)  
4) squareIt(5)
```

```
D:\Python_classes>py test.py  
The Square of 4 is 16  
The Square of 5 is 25
```

Return Statement:

Function can take input values as parameters and executes business logic, and returns output to the caller with return statement.

Q) Write a Function to accept 2 Numbers as Input and return Sum

```
1) def add(x,y):  
2)     return x+y  
3) result=add(10,20)  
4) print("The sum is",result)
```



```
5) print("The sum is",add(100,200))
```

D:\Python_classes>py test.py

The sum is 30

The sum is 300

If we are not writing return statement then default return value is None.

```
1) def f1():
2)     print("Hello")
3) f1()
4) print(f1())
```

Output

Hello

Hello

None

Q) Write a Function to check whether the given Number is Even OR Odd?

```
1) def even_odd(num):
2)     if num%2==0:
3)         print(num,"is Even Number")
4)     else:
5)         print(num,"is Odd Number")
6) even_odd(10)
7) even_odd(15)
```

Output

D:\Python_classes>py test.py

10 is Even Number

15 is Odd Number

Q) Write a Function to find Factorial of given Number?

```
1) def fact(num):
2)     result=1
3)     while num>=1:
4)         result=result*num
5)         num=num-1
6)     return result
7) for i in range(1,5):
```



```
8) print("The Factorial of",i,"is :",fact(i))
```

Output

```
D:\Python_classes>py test.py
The Factorial of 1 is : 1
The Factorial of 2 is : 2
The Factorial of 3 is : 6
The Factorial of 4 is : 24
```

Returning Multiple Values from a Function:

In other languages like C,C++ and Java, function can return atmost one value. But in Python, a function can return any number of values.

Eg 1:

```
1) def sum_sub(a,b):
2)     sum=a+b
3)     sub=a-b
4)     return sum,sub
5) x,y=sum_sub(100,50)
6) print("The Sum is :",x)
7) print("The Subtraction is :",y)
```

Output

```
The Sum is : 150
The Subtraction is : 50
```

Eg 2:

```
1) def calc(a,b):
2)     sum=a+b
3)     sub=a-b
4)     mul=a*b
5)     div=a/b
6)     return sum,sub,mul,div
7) t=calc(100,50)
8) print("The Results are")
9) for i in t:
10)    print(i)
```

Output

```
The Results are
150
```



50
5000
2.0

Types of Arguments

```
def f1(a,b):
```

```
    -----  
    -----  
    -----
```

```
f1(10,20)
```

a, b are formal arguments where as 10,20 are actual arguments.

There are 4 types of arguments allowed in Python.

- 1) Positional Arguments
- 2) Keyword Arguments
- 3) Default Arguments
- 4) Variable Length Arguments

1) Positional Arguments:

- These are the arguments passed to function in correct positional order.

```
def sub(a, b):  
    print(a-b)
```

```
sub(100, 200)  
sub(200, 100)
```

- The number of arguments and position of arguments must be matched. If we change the order then result may be changed.
- If we change the number of arguments then we will get error.

2) Keyword Arguments:

We can pass argument values by keyword i.e by parameter name.

- 1) def wish(name,msg):
- 2) print("Hello",name,msg)
- 3) wish(name="Durga",msg="Good Morning")
- 4) wish(msg="Good Morning",name="Durga")

Output

Hello Durga Good Morning
Hello Durga Good Morning



Here the order of arguments is not important but number of arguments must be matched.

Note: We can use both positional and keyword arguments simultaneously. But first we have to take positional arguments and then keyword arguments, otherwise we will get syntaxerror.

- 1) `def wish(name,msg):`
- 2) `print("Hello",name,msg)`
- 3) `wish("Durga","GoodMorning")` → Valid
- 4) `wish("Durga",msg="GoodMorning")` → Valid
- 5) `wish(name="Durga","GoodMorning")` → Invalid
- 6) `SyntaxError: positional argument follows keyword argument`

3) Default Arguments:

Sometimes we can provide default values for our positional arguments.

- 1) `def wish(name="Guest"):`
- 2) `print("Hello",name,"Good Morning")`
- 3) `wish("Durga")`
- 4) `wish()`

Output

Hello Durga Good Morning

Hello Guest Good Morning

If we are not passing any name then only default value will be considered.

*****Note:**

After default arguments we should not take non default arguments.

- 1) `def wish(name="Guest",msg="Good Morning"):` → Valid
- 2) `def wish(name,msg="Good Morning"):` → Valid
- 3) `def wish(name="Guest",msg):` → Invalid

`SyntaxError: non-default argument follows default argument`

4) Variable Length Arguments:

- Sometimes we can pass variable number of arguments to our function, such type of arguments are called variable length arguments.
- We can declare a variable length argument with * symbol as follows
- `def f1(*n):`
- We can call this function by passing any number of arguments including zero number.
- Internally all these values represented in the form of tuple.



```
1) def sum(*n):
2)     total=0
3)     for n1 in n:
4)         total=total+n1
5)     print("The Sum=",total)
6)
7) sum()
8) sum(10)
9) sum(10,20)
10) sum(10,20,30,40)
```

Output

The Sum= 0

The Sum= 10

The Sum= 30

The Sum= 100

Note: We can mix variable length arguments with positional arguments.

```
1) def f1(n1,*s):
2)     print(n1)
3)     for s1 in s:
4)         print(s1)
5)
6) f1(10)
7) f1(10,20,30,40)
8) f1(10,"A",30,"B")
```

Output

10

10

20

30

40

10

A

30

B

Note: After variable length argument, if we are taking any other arguments then we should provide values as keyword arguments.



```
1) def f1(*s,n1):  
2)     for s1 in s:  
3)         print(s1)  
4)     print(n1)  
5)  
6) f1("A","B",n1=10)
```

Output

A

B

10

f1("A","B",10) → Invalid

TypeError: f1() missing 1 required keyword-only argument: 'n1'

Note: We can declare key word variable length arguments also.

- For this we have to use **.
- def f1(**n):
- We can call this function by passing any number of keyword arguments. Internally these keyword arguments will be stored inside a dictionary.

```
1) def display(**kwargs):  
2)     for k,v in kwargs.items():  
3)         print(k,"=",v)  
4) display(n1=10,n2=20,n3=30)  
5) display(rno=100,name="Durga",marks=70,subject="Java")
```

Output

n1 = 10
n2 = 20
n3 = 30
rno = 100
name = Durga
marks = 70
subject = Java



Case Study:

```
def f(arg1,arg2,arg3=4,arg4=8):
    print(arg1,arg2,arg3,arg4)

1) f(3,2) → 3 2 4 8
2) f(10,20,30,40) → 10 20 30 40
3) f(25,50,arg4=100) → 25 50 4 100
4) f(arg4=2,arg1=3,arg2=4) → 3 4 4 2
```

5) f() → Invalid

TypeError: f() missing 2 required positional arguments: 'arg1' and 'arg2'

6) f(arg3=10, arg4=20, 30, 40) → Invalid

SyntaxError: positional argument follows keyword argument

 [After keyword arguments we should not take positional arguments]

7) f(4, 5, arg2 = 6) → Invalid

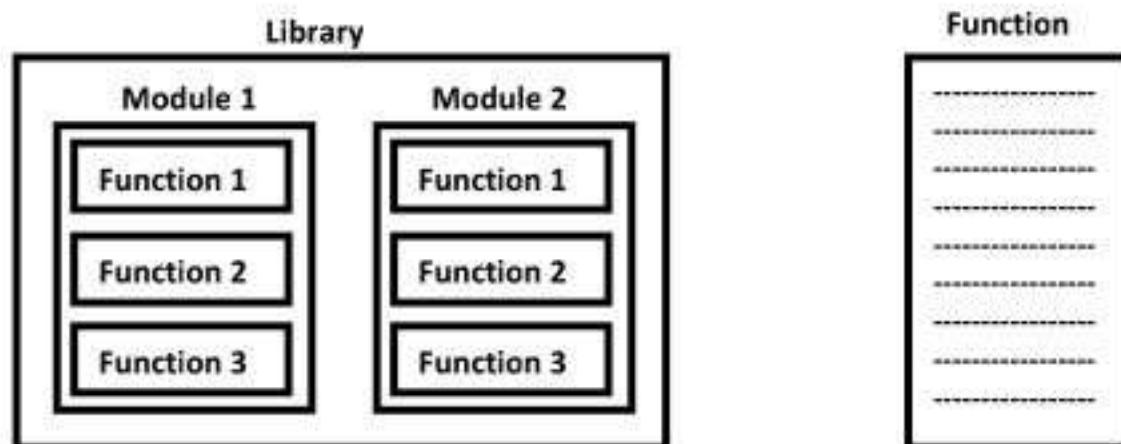
TypeError: f() got multiple values for argument 'arg2'

8) f(4, 5, arg3 = 5, arg5 = 6) → Invalid

TypeError: f() got an unexpected keyword argument 'arg5'

Note: Function vs Module vs Library

- 1) A group of lines with some name is called a function
- 2) A group of functions saved to a file, is called Module
- 3) A group of Modules is nothing but Library





Types of Variables

Python supports 2 types of variables.

- 1) Global Variables
- 2) Local Variables

1) Global Variables

- The variables which are declared outside of function are called global variables.
- These variables can be accessed in all functions of that module.

```
1) a=10 # global variable
2) def f1():
3)     print(a)
4)
5) def f2():
6)     print(a)
7)
8) f1()
9) f2()
```

Output

10
10

2) Local Variables:

- The variables which are declared inside a function are called local variables.
- Local variables are available only for the function in which we declared it.i.e from outside of function we cannot access.

```
1) def f1():
2)     a=10
3)     print(a) # valid
4)
5) def f2():
6)     print(a) #invalid
7)
8) f1()
9) f2()
```

NameError: name 'a' is not defined



global Keyword:

We can use global keyword for the following 2 purposes:

- 1) To declare global variable inside function
- 2) To make global variable available to the function so that we can perform required modifications

```
1) a=10
2) def f1():
3)     a=777
4)     print(a)
5)
6) def f2():
7)     print(a)
8)
9) f1()
10)f2()
11)
```

Output

777

10

```
1) a=10
2) def f1():
3)     global a
4)     a=777
5)     print(a)
6) def f2():
7)     print(a)
8)
9) f1()
10)f2()
```

Output

777

777

```
1) def f1():
2)     a=10
3)     print(a)
4)
5) def f2():
6)     print(a)
7)
```



- 8) f1()
- 9) f2()

Output: NameError: name 'a' is not defined

```
1) def f1():
2)     global a
3)     a=10
4)     print(a)
5)
6) def f2():
7)     print(a)
8)
9) f1()
10) f2()
```

Output

10
10

Note: If global variable and local variable having the same name then we can access global variable inside a function as follows

```
1) a = 10 → Global Variable
2) def f1():
3)     a=777 → Local Variable
4)     print(a)
5)     print(globals()['a'])
6) f1()
```

Output

777
10

Recursive Functions

A function that calls itself is known as Recursive Function.

Eg:

```
factorial(3) = 3 * factorial(2)
              = 3 * 2 * factorial(1)
              = 3 * 2 * 1 * factorial(0)
              = 3 * 2 * 1 * 1
              = 6
```



`factorial(n) = n * factorial(n-1)`

The main advantages of recursive functions are:

- 1) We can reduce length of the code and improves readability.
- 2) We can solve complex problems very easily.

Q) Write a Python Function to find Factorial of given Number with Recursion

```
1) def factorial(n):
2)     if n==0:
3)         result=1
4)     else:
5)         result=n*factorial(n-1)
6)     return result
7) print("Factorial of 4 is :",factorial(4))
8) print("Factorial of 5 is :",factorial(5))
```

Output

Factorial of 4 is : 24

Factorial of 5 is : 120

Anonymous Functions:

- Sometimes we can declare a function without any name, such type of nameless functions are called anonymous functions or lambda functions.
- The main purpose of anonymous function is just for instant use(i.e for one time usage)

Normal Function:

We can define by using def keyword.

```
def squareIt(n):
    return n*n
```

Lambda Function:

We can define by using lambda keyword `lambda n:n*n`

Syntax of lambda Function: `lambda argument_list : expression`

Note: By using Lambda Functions we can write very concise code so that readability of the program will be improved.



Q) Write a Program to create a Lambda Function to find Square of given Number?

```
1) s=lambda n:n*n
2) print("The Square of 4 is :",s(4))
3) print("The Square of 5 is :",s(5))
```

Output

The Square of 4 is : 16

The Square of 5 is : 25

Q) Lambda Function to find Sum of 2 given Numbers

```
1) s=lambda a,b:a+b
2) print("The Sum of 10,20 is:",s(10,20))
3) print("The Sum of 100,200 is:",s(100,200))
```

Output

The Sum of 10,20 is: 30

The Sum of 100,200 is: 300

Q) Lambda Function to find biggest of given Values

```
1) s=lambda a,b:a if a>b else b
2) print("The Biggest of 10,20 is:",s(10,20))
3) print("The Biggest of 100,200 is:",s(100,200))
```

Output

The Biggest of 10,20 is: 20

The Biggest of 100,200 is: 200

Note: Lambda Function internally returns expression value and we are not required to write return statement explicitly.

Note: Sometimes we can pass function as argument to another function. In such cases lambda functions are best choice.

We can use lambda functions very commonly with filter(), map() and reduce() functions, because these functions expect function as argument.



filter() Function:

We can use filter() function to filter values from the given sequence based on some condition.

`filter(function,sequence)`

Where Function Argument is responsible to perform conditional check Sequence can be List OR Tuple OR String.

Q) Program to filter only Even Numbers from the List by using filter() Function?

Without Lambda Function:

```
1) def isEven(x):  
2)     if x%2==0:  
3)         return True  
4)     else:  
5)         return False  
6) l=[0,5,10,15,20,25,30]  
7) l1=list(filter(isEven,l))  
8) print(l1) #[0,10,20,30]
```

With Lambda Function:

```
1) l=[0,5,10,15,20,25,30]  
2) l1=list(filter(lambda x:x%2==0,l))  
3) print(l1) #[0,10,20,30]  
4) l2=list(filter(lambda x:x%2!=0,l))  
5) print(l2) #[5,15,25]
```

map() Function:

- For every element present in the given sequence, apply some functionality and generate new element with the required modification. For this requirement we should go for map() function.
- Eg: For every element present in the list perform double and generate new list of doubles.
- Syntax: `map(function, sequence)`
- The function can be applied on each element of sequence and generates new sequence.



Without Lambda

```
1) l=[1,2,3,4,5]
2) def doubleIt(x):
3)     return 2*x
4) l1=list(map(doubleIt,l))
5) print(l1) #[2, 4, 6, 8, 10]
```

With Lambda

```
1) l=[1,2,3,4,5]
2) l1=list(map(lambda x:2*x,l))
3) print(l1) #[2, 4, 6, 8, 10]
```

Eg 2: To find square of given numbers

```
1) l=[1,2,3,4,5]
2) l1=list(map(lambda x:x*x,l))
3) print(l1) #[1, 4, 9, 16, 25]
```

We can apply map() function on multiple lists also. But make sure all list should have same length.

Syntax: map(lambda x,y:x*y,l1,l2)
x is from l1 and y is from l2

```
1) l1=[1,2,3,4]
2) l2=[2,3,4,5]
3) l3=list(map(lambda x,y:x*y,l1,l2))
4) print(l3) #[2, 6, 12, 20]
```

reduce() Function:

- reduce() function reduces sequence of elements into a single element by applying the specified function.
- reduce(function,sequence)
- reduce() function present in functools module and hence we should write import statement.

```
1) from functools import *
2) l=[10,20,30,40,50]
3) result=reduce(lambda x,y:x+y,l)
4) print(result) # 150
```



Eg:

```
1) result=reduce(lambda x,y:x*y,1)
2) print(result) #12000000
```

Eg:

```
1) from functools import *
2) result=reduce(lambda x,y:x+y,range(1,101))
3) print(result) #5050
```

Everything is an Object:

- In Python every thing is treated as object.
- Even functions also internally treated as objects only.

```
1) def f1():
2)     print("Hello")
3) print(f1)
4) print(id(f1))
```

Output:

<function f1 at 0x00419618>

4298264

Function Aliasing:

For the existing function we can give another name, which is nothing but function aliasing.

```
1) def wish(name):
2)     print("Good Morning:",name)
3)
4) greeting=wish
5) print(id(wish))
6) print(id(greeting))
7)
8) greeting('Durga')
9) wish('Durga')
```

Output:

4429336

4429336

Good Morning: Durga

Good Morning: Durga

**Note:**

- In the above example only one function is available but we can call that function by using either wish name or greeting name.
- If we delete one name still we can access that function by using alias name.

```
1) def wish(name):
2)     print("Good Morning:",name)
3)
4) greeting=wish
5)
6) greeting('Durga')
7) wish('Durga')
8)
9) del wish
10) #wish('Durga') → NameError: name 'wish' is not defined
11) greeting('Pavan')
```

Output:

Good Morning: Durga
Good Morning: Durga
Good Morning: Pavan

Nested Functions:

We can declare a function inside another function, such type of functions are called Nested functions.

```
1) def outer():
2)     print("outer function started")
3)     def inner():
4)         print("inner function execution")
5)     print("outer function calling inner function")
6)     inner()
7) outer()
8) #inner() → NameError: name 'inner' is not defined
```

Output:

outer function started
outer function calling inner function
inner function execution

In the above example inner() function is local to outer() function and hence it is not possible to call directly from outside of outer() function.



Note: A function can return another function.

```
1) def outer():
2)     print("outer function started")
3)     def inner():
4)         print("inner function execution")
5)     print("outer function returning inner function")
6)     return inner
7) f1=outer()
8) f1()
9) f1()
10)f1()
```

Output:

outer function started
outer function returning inner function
inner function execution
inner function execution
inner function execution

Q) What is the difference between the following lines?

```
f1 = outer
f1 = outer()
```

- In the first case for the outer() function we are providing another name f1 (function aliasing).
- But in the second case we calling outer() function, which returns inner function. For that inner function() we are providing another name f1

Note: We can pass function as argument to another function

Eg: filter(function,sequence)
map(function,sequence)
reduce(function,sequence)



- A group of functions, variables and classes saved to a file, which is nothing but module.
- Every Python file (.py) acts as a module.

durgamath.py

```
1) x = 888
2)
3) def add(a,b):
4)     print("The Sum:",a+b)
5)
6) def product(a,b):
7)     print("The Product:",a*b)
```

- durgamath module contains one variable and 2 functions.
- If we want to use members of module in our program then we should import that module.
`import modulename`
- We can access members by using module name.
`modulename.variable`
`modulename.function()`

test.py:

```
1) import durgamath
2) print(durgamath.x)
3) durgamath.add(10,20)
4) durgamath.product(10,20)
```

Output

888
The Sum: 30
The Product: 200

Note: Whenever we are using a module in our program, for that module compiled file will be generated and stored in the hard disk permanently.



Renaming a Module at the time of import **(Module Aliasing):**

- Eg: `import durgamath as m`
- Here `durgamath` is original module name and `m` is alias name.
- We can access members by using alias name `m`

test.py:

- 1) `import durgamath as m`
- 2) `print(m.x)`
- 3) `m.add(10,20)`
- 4) `m.product(10,20)`

from ... import:

We can import particular members of module by using `from ... import`.

The main advantage of this is we can access members directly without using module name.

- 1) `from durgamath import x,add`
- 2) `print(x)`
- 3) `add(10,20)`
- 4) `product(10,20) → NameError: name 'product' is not defined`

We can import all members of a module as follows `from durgamath import *`

test.py:

- 1) `from durgamath import *`
- 2) `print(x)`
- 3) `add(10,20)`
- 4) `product(10,20)`

Various Possibilities of import:

- 1) `import modulename`
- 2) `import module1,module2,module3`
- 3) `import module1 as m`
- 4) `import module1 as m1,module2 as m2,module3`
- 5) `from module import member`
- 6) `from module import member1,member2,memebr3`
- 7) `from module import memeber1 as x`
- 8) `from module import *`



Member Aliasing:

```
1) from durgamath import x as y,add as sum
2) print(y)
3) sum(10,20)
```

Once we defined as alias name,we should use alias name only and we should not use original name

```
1) from durgamath import x as y
2) print(x) → NameError: name 'x' is not defined
```

Reloading a Module:

By default module will be loaded only once even though we are importing multiple times.

module1.py:

```
print("This is from module1")
```

test.py

```
1) import module1
2) import module1
3) import module1
4) import module1
5) print("This is test module")
```

Output

This is from module1

This is test module

- In the above program test module will be loaded only once even though we are importing multiple times.
- The problem in this approach is after loading a module if it is updated outside then updated version of module1 is not available to our program.
- We can solve this problem by reloading module explicitly based on our requirement.
- We can reload by using reload() function of imp module.

```
1) import imp
2) imp.reload(module1)
```

**test.py:**

```
1) import module1
2) import module1
3) from imp import reload
4) reload(module1)
5) reload(module1)
6) reload(module1)
7) print("This is test module")
```

In the above program module1 will be loaded 4 times in that 1 time by default and 3 times explicitly. In this case output is

- 1) This is from module1
- 2) This is from module1
- 3) This is from module1
- 4) This is from module1
- 5) This is test module

The main advantage of explicit module reloading is we can ensure that updated version is always available to our program.

Finding Members of Module by using dir() Function:

Python provides inbuilt function dir() to list out all members of current module or a specified module.

dir() → To list out all members of current module
dir(moduleName) → To list out all members of specified module

Eg 1: test.py

```
1) x=10
2) y=20
3) def f1():
4)     print("Hello")
5) print(dir()) # To print all members of current module
```

Output

```
['__annotations__', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__', 'f1', 'x', 'y']
```