#### Language Fundamentals

Introduction

* Python is a general purpose high level programming language.
* Python was developed by Guido Van Rossam 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**

**3) p s v main(String[] args)**

**5)**

**SOP("Hello world");**

**7) }**

**6) }**

**4) {**

**2) {**

J=

C:

* 1. #include<stdio.h>

**2) void main()**

3) {

**4) print("Hello world");**

5) }

Python:

print("Hello World

**1**

**nd**

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>**

**3) void main()**

**5) int a,b;**

**7) b=20;**

9) **}**

**8) printf("The Sum:%d",(a+b));**

**6) a =10;**

**4) {**

**2)**

Python:

Python:

1) a=10

**2) b=20**

3) print("The Sum:",(a+b))

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+ kerywords 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.

1. 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**

**nd**

1. 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..

1. 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.

1. 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.

1. 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.

1. 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

1. Interpreted:

We are not required to compile Python programs explcitly. 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.

1. Embedded:

We can use Python programs in any other language programs.

i.e we can embedd Python programs anywhere.

1. 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 b'z 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 lanugage Applications

1. Jython or JPython:

It is for Java Applications. It can run on JVM

1. IronPython:

It is for C#.Net platform

1. PyPy:

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

1. RubyPython

For Ruby Platforms

1. 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 

1. Identifier should not starts with digit
   * 123total 
   * total123 √
2. 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 
6. There is no length limit for Python identifiers. But not recommended to use too lengthy identifiers.
7. Dollor ($) Symbol is not allowed in Python.

Q. Which of the following are valid Python identifiers?

1. 123total 
2. total123 √
3. java2share √
4. ca$h 
5. \_abc\_abc\_ √
6. def 
7. if 

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.

Eg: add

### Reserved Words

In Python some words are reserved to represent some meaning or functionality. Such type 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 

a=True √

>>> 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 represent 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

**a**

**a b**

**10**

**a = 10**

**b = 10**

**20**

**a = 10**

**a = 20**

**10**

Note: Python contains several inbuilt functions 1.type()

to check the type of variable

1. id()
2. print()

to print the value

In Python everything is object

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
5. Decimal form(base-10):

It is the default number system in Python The allowed digits are: 0 to 9

Eg: a =10

1. 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

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 Eg:

1) >>> bin(15)

2) '0b1111'

3) >>> bin(0o11)

4) '0b1001'

5) >>> bin(0X10)

6) '0b10000'

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

print(f) 1200.0

instead of 'e' we can use 'E'

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.

Eg:

1) >>> f=0B11.01

2) File "<stdin>", line 1

3) f=0B11.01

**13**

**nd**

|  |
| --- |
| 4) ^ |
| 5) SyntaxError: invalid syntax |
| 6) |
| 7) >>> f=0o123.456 |
| 8) SyntaxError: invalid syntax |
| 9) |
| 10) >>> f=0X123.456 |
| 11) SyntaxError: invalid syntax |

Complex Data Type:

A complex number is of the form



**Real Part Imaginary Part**

**j2 = -1**

**j =**

**a + bj**

a and b contain intergers 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.

1. 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

str 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

**15**

**nd**

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:

slice means a piece

[ ] operator is called slice operator,which can be used to retrieve parts of String. In Python Strings follows zero based index.

The index can be either +ve or -ve.

+ve index means forward direction from Left to Right

-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

**16**

**nd**

|  |
| --- |
| 1) >>> s[1:40] |
| 2) 'urga' |
| 3) >>> s[1:] |
| 4) 'urga' |
| 5) >>> s[:4] |
| 6) 'durg' |
| 7) >>> s[:] |
| 8) 'durga' |
| 9) >>> |
| 10) |
| 11) |
| 12) >>> s\*3 |
| 13) 'durgadurgadurga' |
| 14) |
| 15) >>> len(s) |
| 16) 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.

Eg:

1) >>> c='a'

2) >>> type(c)

3) <class 'str'>

1. long Data Type is available in Python2 but not in Python3. In Python3 long values also we can represent by using int type only.
2. 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 coersion.

The following are inbuilt functions for type casting.

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

1.int():

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

|  |
| --- |
| 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("0B1111") |
| 16) ValueError: invalid literal for int() with base 10: '0B1111' |

Note:

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

2. 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 compulsary str should be either integral or floating point literal and should be specified only in base-10.
3. 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)

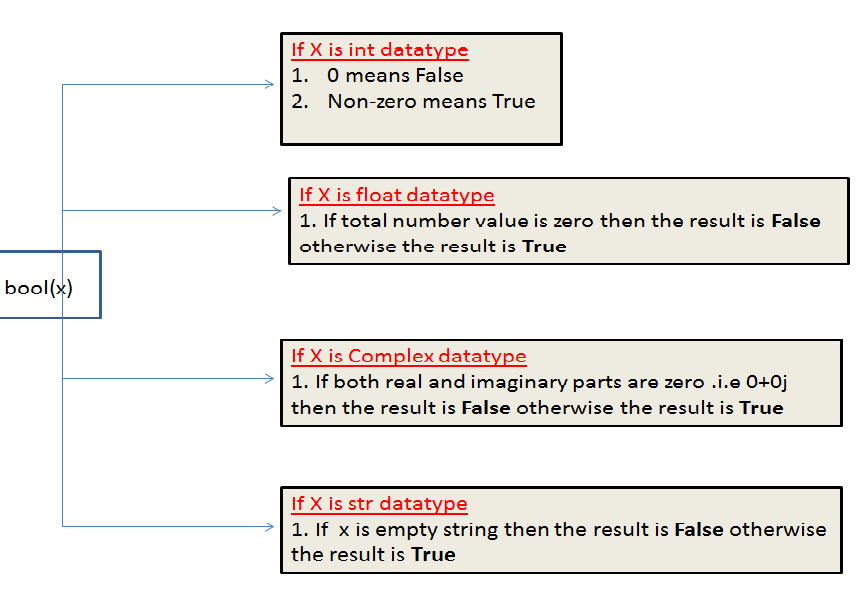
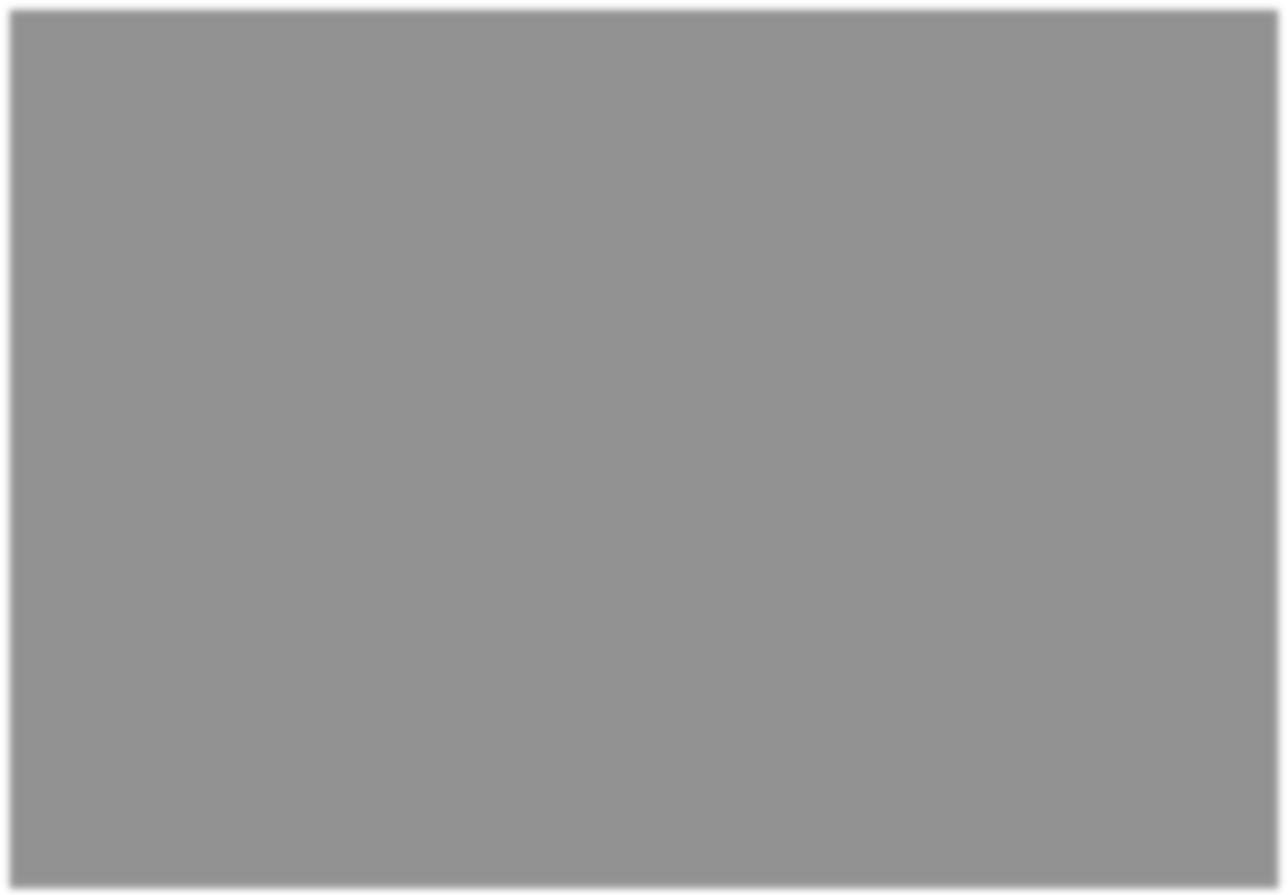
We can use this method to convert x and y into complex number such that x will be real part and y will be imaginary part.

Eg: complex(10,-2)==>10-2j complex(True,False)==>1+0j

1. bool():

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

|  |
| --- |
| 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 |



1. str():

We can use this method to convert other type values to str type Eg:

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-chageable behaviour is called immutability.

In Python if a new object is required, then PVM wont 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.

Eg:

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

bytes Data Type:

bytes data type represens a group of byte numbers just like an array. Eg:

**1) x = [10,20,30,40]**

**3)**

**type(b)==>bytes**

**5)**

**print(b[-1])==> 40**

**7)**

**9)**

**20**

**11)**

**40**

**30**

**10)**

**10**

**8)**

**>>> for i in b : print(i)**

**6)**

**print(b[0])==> 10**

**4)**

**b = bytes(x)**

**2)**

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 |

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)

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]**

**print(list) # [10,10.5,'durga',True,10]**

**2)**

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 eleemnts is nothing but list, where duplicates also allowed.

tuple data type:

tuple data type is exactly same as list data type except that it is immutable.i.e we cannot chage 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

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 r = range(10,20)

for i in r : print(i) 10 to 19

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

2 means increment value 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. 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]

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 |
| 4) |
| 5) set is growable in nature, based on our requirement we can increase or decrease the size. |
| 6) |
| 7) >>> s.add(60) |
| 8) >>> s |
| 9) {0, 100, 'durga', 200, 10, 60} |
| 10) >>> s.remove(100) |
| 11) >>> s |
| 12) {0, 'durga', 200, 10, 60} |

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'

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 wont 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 Python3

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

**29**

**nd**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | <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'> |

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 esacpe 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.

Operators

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 oeprators
5. Assignment operators
6. Special operators

1. Arithmetic Operators:

+ ==>Addition

- ==>Subtraction

\* ==>Multiplication

/ ==>Division operator

% ===>Modulo operator

// ==>Floor Division operator

\*\* ==>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:

|  |  |  |
| --- | --- | --- |
| 1) | Python test.py | or py test.py |
| 2) | a+b= 12 | |
| 3) | a-b= 8 | |
| 4) | a\*b= 20 | |
| 5) | a/b= 5.0 | |
| 6) | a//b= 5 | |
| 7) | a%b= 0 | |
| 8) | 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' |

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

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'

+====>String concatenation operator

\* ===>String multiplication operator

Note: For any number x,

x/0 and x%0 always raises "ZeroDivisionError" 10/0

10.0/0

.....

Relational Operators:

>,>=,<,<= Eg 1:

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") |

Outputa 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

Eg:

|  |  |
| --- | --- |
| 1) | 10<20 ==>True |
| 2) | 10<20<30 ==>True |
| 3) | 10<20<30<40 ==>True |
| 4) | 10<20<30<40>50 ==>False |

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.

Eg:

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

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 evalutates 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

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 repesented directly in the memory where as -ve numbers will be represented indirectly in 2's complement form.

Shift Operators:

<< Left shift operator

After shifting the empty cells we have to fill with zero print(10<<2)==>40

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | **0** | **0** | **0** | **1** | **0** | **1** | **0** |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |

>> 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

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | **0** | **0** | **0** | **1** | **0** | **1** | **0** |

We can apply bitwise operators for boolean types also print(True & False) ==>False

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | **0** | **0** | **0** | **0** | **0** | **1** | **0** |

print(True | False) ===>True print(True ^ False) ==>True print(~True) ==>-2

print(True<<2) ==>4

print(True>>2) ==>0

Assignment Operators:

We can use assignment operator to assign value to the variable.

Eg:

x=10

We can combine asignment 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

Ternary 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

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. 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  Retruns True if the given object not present in the specified Collection Eg:

**1)**

**2)**

**3)**

**4)**

**5)**

**x="hello learning Python is very easy!!!"**

**print('h' in x) True**

**print('d' in x)**

**False**

**print('d' not in x)**

**True**

**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 ()  Parenthesis

\*\*  exponential operator

~,-  Bitwise complement operator,unary minus operator

\*,/,%,//  multiplication,division,modulo,floor division

+,-  addition,subtraction

<<,>>  Left and Right Shift &  bitwise And

^  Bitwise X-OR

|  Bitwise OR

>,>=,<,<=, ==, != ==>Relational or Comparison operators

=,+=,-=,\*=... ==>Assignment operators is , is not  Identity Operators

in , not in  Membership operators not  Logical not

and  Logical and or  Logical or

|  |  |  |
| --- | --- | --- |
| Eg: |  |  |
|  | 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)  10) 3/2\*4+3+(10/5)\*\*3-2  11) 3/2\*4+3+2.0\*\*3-2  12) 3/2\*4+3+8.0-2  13) 1.5\*4+3+8.0-2  14) 6.0+3+8.0-2  15) 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. import math

print(math.sqrt(16))

print(math.pi)

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

import math as m print(m.sqrt(16)) 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.

from math import sqrt,pi print(sqrt(16))

print(pi)

print(math.pi)  NameError: name 'math' is not defined

important functions of math module:

ceil(x) floor(x) pow(x,y) factorial(x) trunc(x) gcd(x,y) sin(x)

cos(x)

tan(x)

....

important variables of math module:

pi3.14 e===>2.71

inf ==>infinity

nan ==>not a number

Q. Write a Python program to find area of circle

pi\*r\*\*2

from math import pi r=16

print("Area of Circle is :",pi\*r\*\*2) OutputArea of Circle is : 804.247719318987

**15**

**nd**

##### Input And Output Statements

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()
3. 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

1. 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 Python3

Eg:

**1) >>> type(input("Enter value:"))**

**3) <class 'str'>**

**5) Enter value:10.5 7)**

**9) <class 'str'>**

**8) Enter value:True**

**6) <class 'str'>**

**4)**

**2) Enter value:10**

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

**1) x=input("Enter First Number:")**

**3) i = int(x)**

**5) print("The Sum:",i+j)**

**7) Enter First Number:100**

**9) The Sum: 300**

**8) Enter Second Number:200**

**6)**

**4) j = int(y)**

**2) y=input("Enter Second Number:")**

-----------------------------------------------------

1. x=int(input("Enter First Number:"))

2) y=int(input("Enter Second Number:"))

3) print("The Sum:",x+y)

-----------------------------------------------------------

1) 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) |
| 12) |

|  |
| --- |
| 13) D:\Python\_classes>py test.py |
| 14) Enter Employee No:100 |
| 15) Enter Employee Name:Sunny |
| 16) Enter Employee Salary:1000 |
| 17) Enter Employee Address:Mumbai |
| 18) Employee Married ?[True|False]:True |
| 19) Please Confirm Information |
| 20) Employee No : 100 |
| 21) Employee Name : Sunny |
| 22) Employee Salary : 1000.0 |
| 23) Employee Address : Mumbai |
| 24) 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) |
| 3) |
| 4) D:\Python\_classes>py test.py |
| 5) Enter 2 numbers :10 20 |
| 6) Product is : 200 |

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

Q. Write a program to read 3 float numbers from the keyboard with , seperator 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) |
| 3) | |
| 4) | D:\Python\_classes>py test.py |
| 5) | Enter 3 float numbers :10.5,20.6,20.1 |
| 6) | 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 Output11.5

eval() can evaluate the Input to list, tuple, set, etc based the provided Input. Eg: Write a Program to accept list from the keynboard on the display

1. l = eval(input(“Enter List”))

2) print (type(l))

3) print(l)

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



Command Line Arguments

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**

**3) print(“The List of Command Line Arguments:”, argv)**

**5) for x in argv:**

**7)**

**9) The Number of Command Line Arguments: 4**

**8) D:\Python\_classes>py test.py 10 20 30**

**print(x)**

**6)**

**4) print(“Command Line Arguments one by one:”)**

**2) print(“The Number of Command Line Arguments:”, len(argv))**

|  |
| --- |
| 10) The List of Command Line Arguments: [‘test.py’, ‘10’,’20’,’30’] |
| 11) Command Line Arguments one by one: |
| 12) test.py |
| 13) 10 |
| 14) 20 |
| 15) 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) |
| 8) |
| 9) D:\Python\_classes>py test.py 10 20 30 40 |
| 10) The Sum: 100 |

Note1: usually space is seperator between command line arguments. If our command line argument itself contains space then we should enclose within double quotes(but not single quotes)

Eg:

**1) from sys import argv 3)**

**5) Sunny**

**7) D:\Python\_classes>py test.py 'Sunny Leone' 9)**

**11) Sunny Leone**

**10) D:\Python\_classes>py test.py "Sunny Leone"**

**8) 'Sunny**

**6)**

**4) D:\Python\_classes>py test.py Sunny Leone**

**2) print(argv[1])**

Note2: 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.

Eg:

1) from sys import argv

2) print(argv[1]+argv[2])

3) print(int(argv[1])+int(argv[2]))

|  |
| --- |
| 4) |
| 5) D:\Python\_classes>py test.py 10 20 |
| 6) 1020 |
| 7) 30 |

Note3: If we are trying to access command line arguments with out of range index then we will get Error.

Eg:

1) from sys import argv

2) print(argv[100])

3)

4) D:\Python\_classes>py test.py 10 20

5) 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 repetetion 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")

3)

4) HelloWorld

5) 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) |
| 3. | |
| 4. | OutputThe Values are : 10 20 30 |

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

**1. a,b,c=10,20,30**

**3. print(a,b,c,sep=':')**

**5. D:\Python\_classes>py test.py 7. 10:20:30**

**6. 10,20,30**

**4.**

**2. print(a,b,c,sep=',')**

Form-4:print() with end attribute:

1. print("Hello")

2. print("Durga")

3. print("Soft")

Output:

1. Hello

2. Durga

3. 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. Eg:

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. Eg:

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:

1) Hello Durga Your Age is 48

2) You are teaching java and Python

Form-7: print(formatted string):

%i====>int

%d====>int

%f=====>float

%s======>String type



Syntax:

print("formatted string" %(variable list)) Eg 1:

**1) a=10**

**3) c=30**

**5) print("b value is %d and c value is %d" %(b,c))**

**7) Output**

**9) b value is 20 and c value is 30**

**8) a value is 10**

**6)**

**4) print("a value is %i" %a)**

**2) b=20**

Eg 2:

**1) s="Durga"**

**3) print("Hello %s ...The List of Items are %s" %(s,list))**

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

**4)**

**2) list=[10,20,30,40]**

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

**1) name="Durga"**

**3) gf="Sunny"**

**5) print("Hello {x} your salary is {y} and Your Friend {z} is waiting".format(x=name,y=salary,z= gf))**

**7) Output**

**9) Hello Durga your salary is 10000 and Your Friend Sunny is waiting**

**8) Hello Durga your salary is 10000 and Your Friend Sunny is waiting**

**6)**

**4) print("Hello {0} your salary is {1} and Your Friend {2} is waiting".format(name,salary,gf))**

**2) salary=10000**



*Flow Control*

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

1. **break**
2. **continue**
3. **pass**

**Transfer Statements**

1. **for**
2. **while**

**Iterative Statements**

**Control Flow**

**Conditional Statements**

1. **if**
2. **if-elif**
3. **if-elif-else**
4. 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:")**

**3)**

**print("Hello Durga Good Morning")**

**5)**

**7) Enter Name:durga**

**9) How are you!!!**

**11) D:\Python\_classes>py test.py**

13) **How are you!!!**

**12) Enter Name:Ravi**

**10)**

**8) Hello Durga Good Morning**

**6) D:\Python\_classes>py test.py**

**4) print("How are you!!!")**

**2) if name=="durga" :**

* 1. 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. Eg:

|  |
| --- |
| 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!!!") |
| 7) |
| 8) D:\Python\_classes>py test.py |
| 9) Enter Name:durga |
| 10) Hello Durga Good Morning |
| 11) How are you!!! |
| 12) |
| 13) D:\Python\_classes>py test.py |
| 14) Enter Name:Ravi |
| 15) Hello Guest Good Moring |
| 16) How are you!!! |



* 1. if-elif-else:

Syntax:

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. Eg:

|  |
| --- |
| 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") |
| 10) |
| 11) |
| 12) D:\Python\_classes>py test.py |
| 13) Enter Your Favourite Brand:RC |
| 14) It is childrens brand |
| 15) |
| 16) D:\Python\_classes>py test.py |
| 17) Enter Your Favourite Brand:KF |
| 18) It is not that much kick |
| 19) |
| 20) D:\Python\_classes>py test.py |
| 21) Enter Your Favourite Brand:KALYANI |
| 22) 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

1. There is no switch statement in Python

Q. Write a program to find biggest of given 2 numbers from the commad 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)**

7)

**8) D:\Python\_classes>py test.py**

9) Enter First Number:10

**10) Enter Second Number:20**

11) Biggest Number is: 20

Q. Write a program to find biggest of given 3 numbers from the commad 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) |
| 10) |
| 11) D:\Python\_classes>py test.py |
| 12) Enter First Number:10 |
| 13) Enter Second Number:20 |
| 14) Enter Third Number:30 |
| 15) Biggest Number is: 30 |
| 16) |
| 17) D:\Python\_classes>py test.py |
| 18) Enter First Number:10 |



19) Enter Second Number:30

**20) Enter Third Number:20**

21) 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 is value in English word?

1. 0==>ZERO

**2) 1 ==>ONE**

3)

**4) n=int(input("Enter a digit from o 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")**

1. 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
3. 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) |
| 4) |
| 5) Output |
| 6) S |
| 7) u |
| 8) n |
| 9) n |
| 10) y |
| 11) |
| 12) L |
| 13) e |
| 14) o |
| 15) n |
| 16) 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 |
| 6) |
| 7) |
| 8) D:\Python\_classes>py test.py |
| 9) Enter some String: Sunny Leone |
| 10) The character present at 0 index is : S |
| 11) The character present at 1 index is : u |
| 12) The character present at 2 index is : n |
| 13) The character present at 3 index is : n |
| 14) The character present at 4 index is : y |
| 15) The character present at 5 index is : |
| 16) The character present at 6 index is : L |
| 17) The character present at 7 index is : e |
| 18) The character present at 8 index is : o |
| 19) The character present at 9 index is : n |
| 20) 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) :**

**print(x)**

**2)**



Eg 7: To print sum of numbers presenst inside list

**1)**

**list=eval(input("Enter List:"))**

**3) for x in list:**

**5) print("The Sum=",sum)**

**7) D:\Python\_classes>py test.py**

**9) The Sum= 100**

**11) D:\Python\_classes>py test.py**

13) **The Sum= 112**

**12) Enter List:[45,67]**

**10)**

**8) Enter List:[10,20,30,40]**

**6)**

**sum=sum+x;**

**4)**

**2) sum=0;**

1. 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. Eg:

|  |
| --- |
| 1) for i in range(4): |
| 2) for j in range(4): |
| 3) print("i=",i," j=",j) |
| 4) |
| 5) Output |
| 6) D:\Python\_classes>py test.py |
| 7) i= 0 j= 0 |
| 8) i= 0 j= 1 |
| 9) i= 0 j= 2 |
| 10) i= 0 j= 3 |
| 11) i= 1 j= 0 |
| 12) i= 1 j= 1 |
| 13) i= 1 j= 2 |
| 14) i= 1 j= 3 |
| 15) i= 2 j= 0 |
| 16) i= 2 j= 1 |
| 17) i= 2 j= 2 |
| 18) i= 2 j= 3 |
| 19) i= 3 j= 0 |
| 20) i= 3 j= 1 |
| 21) i= 3 j= 2 |
| 22) i= 3 j= 3 |



Q. Write a program to dispaly \*'s in Right angled triangled form

**1) \***

**3) \* \* \***

**5) \* \* \* \* \***

**7) \* \* \* \* \* \* \***

**9) n = int(input("Enter number of rows:"))**

**11) for j in range(1,i+1):**

**13) print()**

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

**12)**

**10) for i in range(1,n+1):**

**8)**

**6) \* \* \* \* \* \***

**4) \* \* \* \***

**2) \* \***

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) \* |
| 2) \* \* |
| 3) \* \* \* |
| 4) \* \* \* \* |
| 5) \* \* \* \* \* |
| 6) \* \* \* \* \* \* |
| 7) \* \* \* \* \* \* \* |
| 8) |
| 9) n = int(input("Enter number of rows:")) |
| 10) for i in range(1,n+1): |
| 11) print(" " \* (n-i),end="") |
| 12) print("\* "\*i) |



1. Transfer Statements
   1. break:

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

Eg:

**1) for i in range(10):**

**3)**

**print("processing is enough..plz break")**

**5) print(i)**

**7) D:\Python\_classes>py test.py 9) 1**

**11) 3**

**13) 5**

**15) processing is enough..plz break**

**14) 6**

**12) 4**

**10) 2**

**8) 0**

**6)**

**break**

**4)**

**2) if i==7:**

Eg:

**1) cart=[10,20,600,60,70]**

**3) if item>500:**

**5)**

**break**

**7)**

**9) 10**

**11) To place this order insurence must be required**

**10) 20**

**8) D:\Python\_classes>py test.py**

**6) print(item)**

**print("To place this order insurence must be required")**

**4)**

**2) for item in cart:**



* 1. 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):**

**3)**

**continue**

**5)**

**7) 1**

**9) 5**

11) **9**

**10) 7**

**8) 3**

**6) D:\Python\_classes>py test.py**

**4) print(i)**

**2) if i%2==0:**

Eg 2:

**1) cart=[10,20,500,700,50,60]**

**3) if item>=500:**

**5)**

**continue**

**7)**

**9) D:\Python\_classes>py test.py 11) 20**

**13) We cannot process this item : 700**

15) **60**

**14) 50**

**12) We cannot process this item : 500**

**10) 10**

**8) Output**

**6) print(item)**

**print("We cannot process this item :",item)**

**4)**

**2) for item in cart:**

Eg 3:

**1) numbers=[10,20,0,5,0,30]**

**3) if n==0:**

**5)**

**continue**

**7)**

**6) print("100/{} = {}".format(n,100/n))**

**print("Hey how we can divide with zero..just skipping")**

**4)**

**2) for n in numbers:**



|  |
| --- |
| 8) Output |
| 9) |
| 10) 100/10 = 10.0 |
| 11) 100/20 = 5.0 |
| 12) Hey how we can divide with zero..just skipping |
| 13) 100/5 = 20.0 |
| 14) Hey how we can divide with zero..just skipping |
| 15) 100/30 = 3.3333333333333335 |

loops with else block:

Inside loop execution,if break statement not executed ,then only else part will be executed.

else means loop without break Eg:

|  |
| --- |
| 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") |
| 9) |
| 10) Output |
| 11) 10 |
| 12) 20 |
| 13) 30 |
| 14) 40 |
| 15) 50 |
| 16) 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") |



9)

**10) Output**

11) D:\Python\_classes>py test.py

**12) 10**

13) 20

14) **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

* 1. 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 Eg:

|  |
| --- |
| 1) for i in range(100): |
| 2) if i%9==0: |
| 3) print(i) |
| 4) else:pass |
| 5) |
| 6) D:\Python\_classes>py test.py |
| 7) 0 |
| 8) 9 |
| 9) 18 |
| 10) 27 |
| 11) 36 |
| 12) 45 |
| 13) 54 |
| 14) 63 |
| 15) 72 |
| 16) 81 |
| 17) 90 |
| 18) 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. Eg:

1) x=10

**2) print(x)**

3) del x



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

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.

Eg:

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 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



String Data Type

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 languges 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.

Eg:

s='This is ' single quote symbol' ==>invalid s='This is \' single quote symbol' ==>valid s="This is ' single quote symbol"====>valid s='This is " double quotes symbol' ==>valid

s='The "Python Notes" by 'durga' is very helpful' ==>invalid s="The "Python Notes" by 'durga' is very helpful"==>invalid s='The \"Python Notes\" by \'durga\' is very helpful' ==>valid 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
3. 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' diagram

Eg:

>>> s='durga'

>>> s[0]

'd'

>>> s[4]

'a'

>>> s[-1]

'a'

>>> s[10]

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:")

**4) print("The character present at positive index {} and at nEgative index {} is {}".format(i,i**

**-len(s),x))**

**3) for x in s:**

**2) i=0**

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

1. Accessing characters by using slice operator:

Syntax: s[bEginindex:endindex:step]

bEginindex:From where we have to consider slice(substring) endindex: 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

Eg:

* 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[::]

13) 'Learning Python is very very easy!!!'

14) >>> s[:]

15) 'Learning Python is very very easy!!!'

16) >>> s[::-1]

17) '!!!ysae yrev yrev si nohtyP gninraeL'

Behaviour of slice operator:

s[bEgin:end:step]

step value can be either +ve or –ve

if +ve then it should be forward direction(left to right) and we have to consider bEgin to end-1 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.

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) |
| 6) print("Forward direction") |
| 7) for i in s[::]: |
| 8) print(i,end=' ') |
| 9) |
| 10) print("Backward direction") |
| 11) for i in s[::-1]: |
| 12) 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

Program:

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:durgasoftwaresolutions Enter sub string:durga

durga is found in main string

D:\python\_classes>py test.py

Enter main string:durgasoftwaresolutions 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.

Eg:

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

Eg:

|  |
| --- |
| 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:

find() index()

For backward direction:

rfind() rindex()

1. find():

* 1. ind(substring)

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

Eg:

* + 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

Eg:

|  |  |
| --- | --- |
| 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() method:

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

Eg:

|  |
| --- |
| 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

Eg:

|  |  |
| --- | --- |
| 1) | s="abcabcabcabcadda" |
| 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 oldstring will be replaced with newstring.

Eg1:

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

Output:

Learning Python is very easy

Eg2: 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 happend 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 seperator by using split() method.

l=s.split(seperator)

The default seperator is space. The return type of split() method is List

Eg1:

**1) s="durga software solutions"**

**3) for x in l:**

**print(x)**

**4)**

**2) l=s.split()**

Output: durga software solutions

Eg2:

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 seperator. s=seperator.join(group of strings)

Eg: t=('sunny','bunny','chinny') s='-'.join(t)

print(s)

Output: sunny-bunny-chinny



Eg2:

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

print(s) 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

Eg:

s='learning Python is very Easy' print(s.upper())

print(s.lower()) print(s.swapcase()) print(s.title()) print(s.capitalize())

Output:

LEARNING PYTHON IS VERY EASY

learning python is very easy LEARNING pYTHON IS VERY eASY

Learning Python Is Very Easy 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)

Eg:

s='learning Python is very easy' print(s.startswith('learning')) print(s.endswith('learning')) 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 to9 )
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:

print('Durga786'.isalnum()) #True print('durga786'.isalpha()) #False print('durga'.isalpha()) #True print('durga'.isdigit()) #False print('786786'.isdigit()) #True print('abc'.islower()) #True

print('Abc'.islower()) #False print('abc123'.islower()) #True print('ABC'.isupper()) #True

print('Learning python is Easy'.istitle()) #False print('Learning Python Is Easy'.istitle()) #True 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.

Eg:

name='durga' salary=10000 age=48

print("{} 's salary is {} and his age is {}".format(name,salary,age)) print("{0} 's salary is {1} and his age is {2}".format(name,salary,age)) 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



Important Programs rEgarding String Concept

Q1. Write a program to reverse the given String

input: durga output:agrud

1st Way:

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

2nd Way:

s=input("Enter Some String:") print(''.join(reversed(s)))

3rd Way:

s=input("Enter Some String:") i=len(s)-1

target='' while i>=0:

target=target+s[i] i=i-1

print(target)

Q2. Program to reverse order of words.

|  |
| --- |
| 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=[] |
| 7) i=len(l)-1 |
| 8) while i>=0: |
| 9) l1.append(l[i]) |
| 10) i=i-1 |
| 11) output=' '.join(l1) |
| 12) 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?

1st Way:

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

2nd 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.

s1="ravi" s2="reja"

output: rtaevjia

**16**

**nd**



|  |
| --- |
| 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:**

9) **print(output)**

**output=output+previous\*(int(x)-1)**

**8)**

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:")**

**3) for x in s:**

**5)**

**output=output+x**

**7) else:**

9) **print(output)**

**output=output+chr(ord(previous)+int(x))**

**8)**

**previous=x**

**6)**

**4) if x.isalpha():**

**2) 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: ABCABCABBCDE

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))

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 name='durga'

salary=10000 age=48

print("{} 's salary is {} and his age is {}".format(name,salary,age)) print("{0} 's salary is {1} and his age is {2}".format(name,salary,age)) 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:

print("The intEger number is: {}".format(123)) print("The intEger number is: {:d}".format(123)) print("The intEger number is: {:5d}".format(123)) 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:

print("The float number is: {}".format(123.4567)) print("The float number is: {:f}".format(123.4567))



print("The float number is: {:8.3f}".format(123.4567)) print("The float number is: {:08.3f}".format(123.4567)) print("The float number is: {:08.3f}".format(123.45)) 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 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 print("Binary Form:{0:b}".format(153))

print("Octal Form:{0:o}".format(153)) print("Hexa decimal Form:{0:x}".format(154)) 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:

{:5d} It takes an intEger argument and assigns a minimum width of 5.

{:8.3f} It takes a float argument and assigns a minimum width of 8 including "." and after decimal point excatly 3 digits are allowed with round operation if required

{: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. print("int value with sign:{:+d}".format(123))

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

Eg:

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 convinient way is to use \*\*person

Eg:

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()

Eg:

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

Eg:

**1) class Person:**

**3)**

**self.name=name**

**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)))**

**self.age=age**

**4)**

**2) def init (self,name,age):**

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



List Data Structure

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 seperator.

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=[]**

**3) print(type(list)) 5) []**

**6) <class 'list'>**

**4)**

**2) print(list)**

1. If we know elements already then we can create list as follows list=[10,20,30,40]



1. With dynamic input:

|  |  |
| --- | --- |
| 1) | list=eval(input("Enter List:")) |
| 2) | print(list) |
| 3) | print(type(list)) |
| 4) | |
| 5) | D:\Python\_classes>py test.py |
| 6) | Enter List:[10,20,30,40] |
| 7) | [10, 20, 30, 40] |
| 8) | <class 'list'> |

1. With list() function:

**1) l=list(range(0,10,2))**

**3) print(type(l))**

**5) D:\Python\_classes>py test.py**

**7) <class 'list'>**

**6) [0, 2, 4, 6, 8]**

**4)**

**2) print(l)**

Eg:

|  |  |
| --- | --- |
| 1) | s="durga" |
| 2) | l=list(s) |
| 3) | print(l) |
| 4) | |
| 5) | D:\Python\_classes>py test.py |
| 6) | ['d', 'u', 'r', 'g', 'a'] |

1. with split() function:

|  |  |
| --- | --- |
| 1) | s="Learning Python is very very easy !!!" |
| 2) | l=s.split() |
| 3) | print(l) |
| 4) | print(type(l)) |
| 5) | |
| 6) | D:\Python\_classes>py test.py |
| 7) | ['Learning', 'Python', 'is', 'very', 'very', 'easy', '!!!'] |
| 8) | <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]

|  |  |  |  |
| --- | --- | --- | --- |
| **-4** | **-3** | **-2 -1** | |
| **10** | **20** | **30** | **40** |
| **0** | **1** | **2 3** | |

list

print(list[0]) ==>10

print(list[-1]) ==>40

print(list[10]) ==>IndexError: list index out of range

1. 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

Eg:

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])



|  |
| --- |
| 7) |
| 8) Output |
| 9) D:\Python\_classes>py test.py |
| 10) [3, 5, 7] |
| 11) [5, 7, 9] |
| 12) [4, 5, 6, 7] |
| 13) [9, 7, 5] |
| 14) [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. Eg:

|  |  |
| --- | --- |
| 1) | n=[10,20,30,40] |
| 2) | print(n) |
| 3) | n[1]=777 |
| 4) | print(n) |
| 5) | |
| 6) | D:\Python\_classes>py test.py |
| 7) | [10, 20, 30, 40] |
| 8) | [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]**

**3) while i<len(n):**

**5) i=i+1**

**7) D:\Python\_classes>py test.py 9) 1**

**11) 3**

**13) 5**

**15) 7**

**17) 9**

**16) 8**

**14) 6**

**12) 4**

**10) 2**

**8) 0**

**6)**

**4) print(n[i])**

**2) i=0**



18) 10

1. By using for loop:

**1) n=[0,1,2,3,4,5,6,7,8,9,10]**

**3) print(n1)**

**5) D:\Python\_classes>py test.py 7) 1**

**9) 3**

**11) 5**

**13) 7**

**15) 9**

**16) 10**

**14) 8**

**12) 6**

**10) 4**

**8) 2**

**6) 0**

**4)**

**2) for n1 in n:**

1. To display only even numbers:

**1) n=[0,1,2,3,4,5,6,7,8,9,10]**

**3) if n1%2==0:**

**5)**

**7) 0**

**9) 4**

**11) 8**

12) **10**

**10) 6**

**8) 2**

**6) D:\Python\_classes>py test.py**

**print(n1)**

**4)**

**2) for n1 in n:**

1. 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)

5)

6) Output

7) D:\Python\_classes>py test.py

8) A is available at positive index: 0 and at negative index: -3

9) B is available at positive index: 1 and at negative index: -2

10) C is available at positive index: 2 and at negative index: -1



Important functions of List:

* 1. 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

1. 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)) |
| 6) |
| 7) Output |
| 8) D:\Python\_classes>py test.py |
| 9) 1 |
| 10) 4 |
| 11) 2 |
| 12) 0 |

1. index() function:

returns the index of first occurrence of the specified item. Eg:

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



* 1. Manipulating elements of List:
     1. append() function:

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

|  |
| --- |
| 1) list=[] |
| 2) list.append("A") |
| 3) list.append("B") |
| 4) list.append("C") |
| 5) print(list) |
| 6) |
| 7) D:\Python\_classes>py test.py |
| 8) ['A', 'B', 'C'] |

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

**1) list=[]**

**3) if i%10==0:**

**5) print(list) 7)**

**9) [0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]**

**8) D:\Python\_classes>py test.py**

**6)**

**list.append(i)**

**4)**

**2) for i in range(101):**

* + 1. insert() function:

To insert item at specified index position

|  |  |
| --- | --- |
| 1) | n=[1,2,3,4,5] |
| 2) | n.insert(1,888) |
| 3) | print(n) |
| 4) | |
| 5) | D:\Python\_classes>py test.py |
| 6) | [1, 888, 2, 3, 4, 5] |

Eg:

1) n=[1,2,3,4,5]

2) n.insert(10,777)

3) n.insert(-10,999)

4) print(n)

5)



6) D:\Python\_classes>py test.py

7) [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 |

* + 1. 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 Eg:

**1)**

**2)**

**3)**

**4)**

**5)**

**6)**

**7)**

**order1=["Chicken","Mutton","Fish"]**

**order2=["RC","KF","FO"]**

**order1.extend(order2) print(order1)**

**D:\Python\_classes>py test.py**

**['Chicken', 'Mutton', 'Fish', 'RC', 'KF', 'FO']**

Eg:

**1)**

**2)**

**3)**

**4)**

**5)**

**6)**

**order=["Chicken","Mutton","Fish"]**

**order.extend("Mushroom") print(order)**

**D:\Python\_classes>py test.py**

**['Chicken', 'Mutton', 'Fish', 'M', 'u', 's', 'h', 'r', 'o', 'o', 'm']**

* + 1. 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) |
| 4) | |
| 5) | D:\Python\_classes>py test.py |
| 6) | [20, 10, 30] |

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

**1) n=[10,20,10,30]**

**3) print(n)**

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

**4)**

**2) n.remove(40)**

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

* + 1. pop() function:

It removes and returns the last element of the list.

This is only function which manipulates list and returns some element. Eg:

1) n=[10,20,30,40]

2) print(n.pop())

3) print(n.pop())

4) print(n)

5)

6) D:\Python\_classes>py test.py

7) 40

8) 30

9) [10, 20]

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

1) n=[]

**9**

**nd**

2) print(n.pop()) ==> IndexError: pop from empty list



**10**

**nd**

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



* 1. 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) |
| 4) | |
| 5) | D:\Python\_classes>py test.py |
| 6) | [40, 30, 20, 10] |

1. sort() function:

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]**

**3) print(n) #[0,5,10,15,20]**

**5) s=["Dog","Banana","Cat","Apple"]**

7) **print(s) #['Apple','Banana','Cat','Dog']**

**6) s.sort()**

**4)**

**2) n.sort()**

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

Eg:

**1) n=[20,10,"A","B"]**

**3) print(n)**

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

**4)**

**2) n.sort()**

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.

Eg:

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. Eg:

**1) x=[10,20,30,40]**

**3) print(id(x))**

**x**

**y**

4) **print(id(y))**

**40**

**30**

**20**

**10**

**2) y=x**

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 | 10 | 20 | 30 | 40 |  |
| 3) y[1]=777 x |  | 777 |  |  |  |
| 4) print(x) ==>[10,777,30,40] y | | | | | |

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]

x

|  |  |  |  |
| --- | --- | --- | --- |
| **10** | **20** | **30** | **40** |

|  |  |  |  |
| --- | --- | --- | --- |
| 10 | 20  777 | 30 | 40 |

y

1. 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]

x

|  |  |  |  |
| --- | --- | --- | --- |
| **10** | **20** | **30** | **40** |

|  |  |  |  |
| --- | --- | --- | --- |
| 10 | 20  777 | 30 | 40 |

y

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

1. Repetition Operator(\*):

We can use repetition operator \* to repeat elements of list specified number of times Eg:

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. Eg:

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 relatational operators(<,<=,>,>=) between List objects,only first element comparison will be performed.

Eg:

|  |  |  |
| --- | --- | --- |
| 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 memebership operators.

in operator not in operator

Eg:

**1. n=[10,20,30,40]**

**3. print (10 not in n)**

**5. print (50 not in n)**

**7. Output**

**6.**

**4. print (50 in n)**

**2. print (10 in n)**



|  |
| --- |
| 8. True |
| 9. False |
| 10. False |
| 11. True |

clear() function:

We can use clear() function to remove all elements of List. Eg:

1. n=[10,20,30,40]

2. print(n)

3. n.clear()

4. print(n)

5.

6. Output

7. D:\Python\_classes>py test.py

8. [10, 20, 30, 40]

9. []

Nested Lists:

Sometimes we can take one list inside another list. Such type of lists are called nested lists.

Eg:

|  |
| --- |
| 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]) |
| 7. |
| 8. Output |
| 9. D:\Python\_classes>py test.py |
| 10. [10, 20, [30, 40]] |
| 11. 10 |
| 12. [30, 40] |
| 13. 30 |
| 14. 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() |
| 11) |
| 12) Output |
| 13) D:\Python\_classes>py test.py |
| 14) [[10, 20, 30], [40, 50, 60], [70, 80, 90]] |
| 15) Elements by Row wise: |
| 16) [10, 20, 30] |
| 17) [40, 50, 60] |
| 18) [70, 80, 90] |
| 19) Elements by Matrix style: |
| 20) 10 20 30 |
| 21) 40 50 60 |
| 22) 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] Eg:

|  |
| --- |
| 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) |
| 7) |
| 8) Output |
| 9) D:\Python\_classes>py test.py |
| 10) [1, 4, 9, 16, 25, 36, 49, 64, 81, 100] |



11) [2, 4, 8, 16, 32]

12) [4, 16, 36, 64, 100]

Eg:

**1) words=["Balaiah","Nag","Venkatesh","Chiranjeevi"]**

**3) print(l)**

**5) Output['B', 'N', 'V', 'C']**

**4)**

**2) l=[w[0] for w in words]**

Eg:

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)

5)

6) Output

7) ['the', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog']

8) [['THE', 3], ['QUICK', 5], ['BROWN', 5], ['FOX', 3], ['JUMPS', 5], ['OVER', 4],

9) ['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']**

**3) found=[]**

**5) if letter in vowels:**

**7)**

**found.append(letter)**

**9) print("The number of different vowels present in",word,"is",len(found))**

**11)**

**12) D:\Python\_classes>py test.py**

**10)**

**8) print(found)**

**if letter not in found:**

**6)**

**4) for letter in word:**

**2) word=input("Enter the word to search for vowels: ")**



13) Enter the word to search for vowels: durgasoftwaresolutions

14) ['u', 'a', 'o', 'e', 'i']

15) The number of different vowels present in durgasoftwaresolutions is 5

list out all functions of list and write a program to use these functions



### Tuple Data Structure

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.

Hence Tuple is Read Only version of List.

1. If our data is fixed and never changes then we should go for Tuple.
2. Insertion Order is preserved
3. Duplicates are allowed
4. Heterogeneous objects are allowed.
5. We can preserve insertion order and we can differentiate duplicate objects by using index. Hence index will play very important role in Tuple also.

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)

1. We can represent Tuple elements within Parenthesis and with comma seperator. Parenethesis are optional but recommended to use.

Eg:

|  |
| --- |
| 1. t=10,20,30,40 |
| 2. print(t) |
| 3. print(type(t)) |
| 4. |
| 5. Output |
| 6. (10, 20, 30, 40) |
| 7. <class 'tuple'> |
| 8. |
| 9. t=() |
| 10. print(type(t)) # tuple |

Note: We have to take special care about single valued tuple.compulsary the value should ends with comma,otherwise it is not treated as tuple.

Eg:

**1. t=(10)**

**3. print(type(t))**

**5. Output**

**7. <class 'int'>**

**6. 10**

**4.**

**2. print(t)**



Eg:

**1. t=(10,)**

**3. print(type(t))**

**5. Output**

**7. <class 'tuple'>**

**6. (10,)**

**4.**

**2. print(t)**

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 |

1. 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])

5.

6. Output

7. (30, 40, 50)

8. (30, 40, 50, 60)

9. (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) |



1. 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

1. count()

To return number of occurrences of given element in the tuple

Eg:

t=(10,20,10,10,20)

print(t.count(10)) #3

1. 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

1. sorted()

To sort elements based on default natural sorting order

1. t=(40,10,30,20)

2. t1=sorted(t)

* 1. print(t1)

4. print(t)

5.

6. Output

7. [10, 20, 30, 40]

8. (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]

1. min() and max() functions:

These functions return min and max values according to default natural sorting order. Eg:

1. t=(40,10,30,20)

2. print(min(t)) #10

* 1. print(max(t)) #40

1. 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 Eg:

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 Eg:

|  |
| --- |
| 1. t=(10,20,30,40) |
| 2. a,b,c,d=t |
| 3. print("a=",a,"b=",b,"c=",c,"d=",d) |
| 4. |
| 5. Output |
| 6. 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))**

**3. for x in t:**

**5.**

**7. D:\Python\_classes>py test.py 9. 1**

**11. 9**

**13. 25**

**12. 16**

**10. 4**

**8. <class 'generator'>**

**6. Output**

**4. print(x)**

**2. print(type(t))**



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:"))**

**3. sum=0**

**5. sum=sum+x**

**7. print("The Average=",sum/l)**

**9. D:\Python\_classes>py test.py**

**11. The Sum= 100 13.**

**15. Enter Tuple of Numbers:(100,200,300)**

**17. The Average= 200.0**

**16. The Sum= 600**

**14. D:\Python\_classes>py test.py**

**12. The Average= 25.0**

**10. Enter Tuple of Numbers:(10,20,30,40)**

**8.**

**6. print("The Sum=",sum)**

**4. for x in t:**

**2. l=len(t)**

Differences between List and Tuple:

List and Tuple are exactly same except small difference: List objects are mutable where as Tuple objects are immutable.

In both cases insertion order is preserved, duplicate objects are allowed, heterogenous objects are allowed, index and slicing are supported.

|  |  |
| --- | --- |
| List | Tuple |
| 1) List is a Group of Comma separeated Values within Square Brackets and Square Brackets are mandatory.  Eg: i = [10, 20, 30, 40] | 1) Tuple is a Group of Comma separeated Values within Parenthesis and Parenthesis are optional.  Eg: t = (10, 20, 30, 40)  t = 10, 20, 30, 40 |
| 2) List Objects are Mutable i.e. once we creates List Object we can perform any changes in that Object.  Eg: i[1] = 70 | 2) Tuple Objeccts are Immutable i.e. once we creates Tuple Object we cannot change its content.  t[1] = 70  ValueError: tuple object does not support item assignment. |
| 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 Dictionries because Keys should be  Hashable and Immutable. | 4) Tuple Objects can be used as Keys for Dictionries because Keys should be  Hashable and Immutable. |





### Set Data Structure

* 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 seperation
* We can apply mathematical operations like union,intersection,difference etc on set objects.

Creation of Set objects:

Eg:

**1. s={10,20,30,40}**

**3. print(type(s))**

**5. Output**

**7. <class 'set'>**

**6. {40, 10, 20, 30}**

**4.**

**2. print(s)**

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.

|  |  |  |
| --- | --- | --- |
| Eg: |  | |
|  | 1. | s={} |
|  | 2. | print(s) |
|  | 3. | print(type(s)) |
|  | 4. |  |
|  | 5. | Output |
|  | 6. | {} |
|  | 7. | <class 'dict'> |
| Eg: |  |  |
|  | 1. | s=set() |
|  | 2. | print(s) |
|  | 3. | print(type(s)) |
|  | 4. |  |
|  | 5. | Output |
|  | 6. | set() |
|  | 7. | <class 'set'> |

Important functions of set:

1. add(x):

Adds item x to the set Eg:

1. s={10,20,30}

2. s.add(40);

3. print(s) #{40, 10, 20, 30}

1. 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.

Eg:

|  |  |
| --- | --- |
| 1. | s={10,20,30} |
| 2. | l=[40,50,60,10] |
| 3. | s.update(l,range(5)) |
| 4. | print(s) |



5.

6. Output

7. {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)

1. s.add(10,20,30) TypeError: add() takes exactly one argument (3 given)
2. 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.

s={10,20,30}

s1=s.copy() print(s1)

1. pop():

It removes and returns some random element from the set. Eg:

1. s={40,10,30,20}

2. print(s)

* 1. print(s.pop())

4. print(s)

5.

6. Output

7. {40, 10, 20, 30}

8. 40

9. {10, 20, 30}



1. remove(x):

It removes specified element from the set.

If the specified element not present in the Set then we will get KeyError

s={40,10,30,20}

s.remove(30)

print(s) # {40, 10, 20}

s.remove(50) ==>KeyError: 50

1. discard(x):

It removes the specified element from the set.

If the specified element not present in the set then we won't get any error.

s={10,20,30}

s.discard(10)

print(s) ===>{20, 30} s.discard(50)

print(s) ==>{20, 30}

Q. What is the difference between remove() and discard() functions in Set?

Q. Explain differences between pop(),remove() and discard() functionsin Set?

1. clear():

To remove all elements from the Set.

1. s={10,20,30}

2. print(s)

* 1. s.clear()

4. print(s)

5.

6. Output

7. {10, 20, 30}

8. 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

Eg:

x={10,20,30,40}

y={30,40,50,60}

print(x.union(y)) #{10, 20, 30, 40, 50, 60}

print(x|y) #{10, 20, 30, 40, 50, 60}

1. intersection():

x.intersection(y) or x&y

Returns common elements present in both x and y Eg:

x={10,20,30,40}

y={30,40,50,60}

print(x.intersection(y)) #{40, 30}

print(x&y) #{40, 30}

1. difference():

x.difference(y) or x-y

returns the elements present in x but not in y Eg:

x={10,20,30,40}

y={30,40,50,60}

print(x.difference(y)) #{10, 20}

print(x-y) #{10, 20}

print(y-x) #{50, 60}



1. symmetric\_difference():
2. symmetric\_difference(y) or x^y

Returns elements present in either x or y but not in both Eg:

x={10,20,30,40}

y={30,40,50,60}

print(x.symmetric\_difference(y)) #{10, 50, 20, 60}

print(x^y) #{10, 50, 20, 60}

Membership operators: (in , not in)

Eg:

* 1. s=set("durga")

2. print(s)

3. print('d' in s)

4. print('z' in s)

5.

6. Output

7. {'u', 'g', 'r', 'd', 'a'}

8. True

9. False

Set Comprehension:

Set comprehension is possible.

s={x\*x for x in range(5)} print(s) #{0, 1, 4, 9, 16}

s={2\*\*x for x in range(2,10,2)} print(s) #{16, 256, 64, 4}

set objects won't support indexing and slicing:

Eg:

s={10,20,30,40}

print(s[0]) ==>TypeError: 'set' object does not support indexing 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) |
| 4. | |
| 5. | Output |
| 6. | D:\Python\_classes>py test.py |
| 7. | Enter List of values: [10,20,30,10,20,40] |
| 8. | {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)

7.

8. Output

9. D:\Python\_classes>py test.py

10. Enter List of values: [10,20,30,10,20,40]

11. [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) |
| 6. |
| 7. Output |
| 8. D:\Python\_classes>py test.py |
| 9. Enter word to search for vowels: durga |
| 10. The different vowel present in durga are {'u', 'a'} |



Dictionary Data Structure

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

d[100]="durga" d[200]="ravi" d[300]="shiva"

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.

d={100:'durga' ,200:'ravi', 300:'shiva'} print(d[100]) #durga

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]) |
| 12) |
| 13) Output |
| 14) D:\Python\_classes>py test.py |
| 15) Enter number of students: 3 |
| 16) Enter Student Name: durga |
| 17) Enter % of Marks of Student: 60% |
| 18) Enter Student Name: ravi |
| 19) Enter % of Marks of Student: 70% |
| 20) Enter Student Name: shiva |



|  |  |
| --- | --- |
| 1. Enter % of Marks of Student: 80% 2. Name of Student % of marks | |
| 23) durga | 60% |
| 24) ravi | 70 % |
| 25) 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. Eg:

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)

7.

8. Output

9. {100: 'durga', 200: 'ravi', 300: 'shiva'}

10. {100: 'durga', 200: 'ravi', 300: 'shiva', 400: 'pavan'}

11. {100: 'sunny', 200: 'ravi', 300: 'shiva', 400: 'pavan'}

How to delete elements from dictionary?

del d[key]

It deletes entry associated with the specified key. If the key is not available then we will get KeyError

Eg:

|  |
| --- |
| 1. d={100:"durga",200:"ravi",300:"shiva"} |
| 2. print(d) |
| 3. del d[100] |
| 4. print(d) |
| 5. del d[400] |
| 6. |
| 7. Output |
| 8. {100: 'durga', 200: 'ravi', 300: 'shiva'} |



9. {200: 'ravi', 300: 'shiva'}

10. **KeyError: 400**

d.clear()

To remove all entries from the dictionary Eg:

1. d={100:"durga",200:"ravi",300:"shiva"}

2. print(d)

3. d.clear()

4. print(d)

5.

6. Output

7. {100: 'durga', 200: 'ravi', 300: 'shiva'}

8. **{}**

del d

To delete total dictionary.Now we cannot access d Eg:

1. d={100:"durga",200:"ravi",300:"shiva"}

2. print(d)

3. del d

4. print(d)

5.

6. Output

7. {100: 'durga', 200: 'ravi', 300: 'shiva'}

8. 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



1. len()

Returns the number of items in the dictionary

1. clear():

To remove all elements from the dictionary

1. 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.

Eg:

d={100:"durga",200:"ravi",300:"shiva"} print(d[100]) ==>durga

print(d[400]) ==>KeyError:400 print(d.get(100)) ==durga print(d.get(400)) ==>None print(d.get(100,"Guest")) ==durga print(d.get(400,"Guest")) ==>Guest

1. 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 Eg:

1) d={100:"durga",200:"ravi",300:"shiva"}

2) print(d.pop(100))

3) print(d)

4) print(d.pop(400))

5)

6) Output



7) durga

8) {200: 'ravi', 300: 'shiva'}

9) KeyError: 400

1. popitem():

It removes an arbitrary item(key-value) from the dictionaty and returns it. Eg:

1) d={100:"durga",200:"ravi",300:"shiva"}

2) print(d)

3) print(d.popitem())

4) print(d)

5)

6) Output

7) {100: 'durga', 200: 'ravi', 300: 'shiva'}

8) (300, 'shiva')

9) {100: 'durga', 200: 'ravi'}

If the dictionary is empty then we will get KeyError d={}

print(d.popitem()) ==>KeyError: 'popitem(): dictionary is empty'

1. keys():

It returns all keys associated eith dictionary Eg:

1) d={100:"durga",200:"ravi",300:"shiva"}

2) print(d.keys())

3) for k in d.keys():

4) print(k)

5)

6) Output

7) dict\_keys([100, 200, 300])

8) 100

9) 200

10) 300

1. values():

It returns all values associated with the dictionary



Eg:

|  |
| --- |
| 1. d={100:"durga",200:"ravi",300:"shiva"} |
| 2. print(d.values()) |
| 3. for v in d.values(): |
| 4. print(v) |
| 5. |
| 6. Output |
| 7. dict\_values(['durga', 'ravi', 'shiva']) |
| 8. durga |
| 9. ravi |
| 10. shiva |

1. items():

It returns list of tuples representing key-value pairs. [(k,v),(k,v),(k,v)]

Eg:

|  |  |
| --- | --- |
| 1. | d={100:"durga",200:"ravi",300:"shiva"} |
| 2. | for k,v in d.items(): |
| 3. | print(k,"--",v) |
| 4. | |
| 5. | Output |
| 6. | 100 -- durga |
| 7. | 200 -- ravi |
| 8. | 300 -- shiva |

1. copy():

To create exactly duplicate dictionary(cloned copy) d1=d.copy();

1. 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.

Eg:

**1. d={100:"durga",200:"ravi",300:"shiva"}**

**3. print(d)**

**5. print(d)**

**7. Output**

**9. {100: 'durga', 200: 'ravi', 300: 'shiva', 400: 'pavan'}**

**11. {100: 'durga', 200: 'ravi', 300: 'shiva', 400: 'pavan'}**

**10. durga**

**8. pavan**

**6.**

**4. print(d.setdefault(100,"sachin"))**

**2. print(d.setdefault(400,"pavan"))**

1. 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) |
| 4. | |
| 5. | Output |
| 6. | D:\Python\_classes>py test.py |
| 7. | Enter dictionary:{'A':100,'B':200,'C':300} |
| 8. | 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")

7.

8. Output

9. D:\Python\_classes>py test.py

10. Enter any word: mississippi

11. m occurred 1 times

12. i occurred 4 times

13. s occurred 4 times

14. 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: ") 3. d={}**

**5. if x in vowels:**

**7. for k,v in sorted(d.items()): 9.**

**11. D:\Python\_classes>py test.py**

**13. a occurred 4 times**

**15. o occurred 2 times**

**14. i occurred 2 times**

**12. Enter any word: doganimaldoganimal**

**10. Output**

**8. print(k,"occurred ",v," times")**

**d[x]=d.get(x,0)+1**

**6.**

**4. for x in word:**

**2. vowels={'a','e','i','o','u'}**

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: "))**

**3) for i in range(n):**

**5) marks=input("Enter Student Marks: ")**

**7) while True:**

**9) marks=d.get(name,-1)**

**11)**

**print("Student Not Found")**

**13)**

**print("The Marks of",name,"are",marks)**

**15) if option=="No":**

**17) print("Thanks for using our application")**

**19) Output**

**21) Enter the number of students: 5**

**23) Enter Student Marks: 90**

**22) Enter Student Name: sunny**

**20) D:\Python\_classes>py test.py**

**18)**

**break**

**16)**

**14) option=input("Do you want to find another student marks[Yes|No]")**

**12) else:**

**10) if marks== -1:**

**8) name=input("Enter Student Name to get Marks: ")**

**6) d[name]=marks**

**4) name=input("Enter Student Name: ")**

**2) d={}**

|  |
| --- |
| 24) Enter Student Name: banny |
| 25) Enter Student Marks: 80 |
| 26) Enter Student Name: chinny |
| 27) Enter Student Marks: 70 |
| 28) Enter Student Name: pinny |
| 29) Enter Student Marks: 60 |
| 30) Enter Student Name: vinny |
| 31) Enter Student Marks: 50 |
| 32) Enter Student Name to get Marks: sunny |
| 33) The Marks of sunny are 90 |
| 34) Do you want to find another student marks[Yes|No]Yes |
| 35) Enter Student Name to get Marks: durga |
| 36) Student Not Found |
| 37) Do you want to find another student marks[Yes|No]No |
| 38) 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) |
| 5. |
| 6. Output |
| 7. {1: 1, 2: 4, 3: 9, 4: 16, 5: 25} |
| 8. {1: 2, 2: 4, 3: 6, 4: 8, 5: 10} |

FUNCTIONS

If a group of statements is repeatedly required then it is not recommended to write these statements everytime seperately.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
3. 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..

1. 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):**

**3. wish("Durga") 5.**

**7. D:\Python\_classes>py test.py**

9. **Hello Ravi Good Morning**

**8. Hello Durga Good Morning**

**6.**

**4. wish("Ravi")**

**print("Hello",name," Good Morning")**

**2.**

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) |
| 5. | |
| 6. | D:\Python\_classes>py test.py |
| 7. | The Square of 4 is 16 |
| 8. | 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)) |
| 6. |
| 7. |
| 8. D:\Python\_classes>py test.py |
| 9. The sum is 30 |
| 10. The sum is 300 |

If we are not writing return statement then default return value is None

|  |  |  |
| --- | --- | --- |
| Eg: |  | |
|  | 1. | def f1(): |
|  | 2. | print("Hello") |
|  | 3. | f1() |
|  | 4. | print(f1()) |
|  | 5. |  |
|  | 6. | Output |
|  | 7. | Hello |
|  | 8. | Hello |
|  | 9. | 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) |
| 8. |
| 9. Output |
| 10. D:\Python\_classes>py test.py |
| 11. 10 is Even Number |
| 12. 15 is Odd Number |

Q. Write a function to find factorial of given number?

**1) def fact(num):**

**3) while num>=1:**

**5)**

**num=num-1**

**7) for i in range(1,5):**

**9)**

**11) D:\Python\_classes>py test.py**

**13) The Factorial of 2 is : 2**

**15) The Factorial of 4 is : 24**

**14) The Factorial of 3 is : 6**

**12) The Factorial of 1 is : 1**

**10) Output**

**8) print("The Factorial of",i,"is :",fact(i))**

**6) return result**

**result=result\*num**

**4)**

**2) result=1**

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):**

**3) sub=a-b**

**5) x,y=sum\_sub(100,50)**

**7) print("The Subtraction is :",y)**

**9) Output**

11) **The Subtraction is : 50**

**10) The Sum is : 150**

**8)**

**6) print("The Sum is :",x)**

**4) return sum,sub**

**2) sum=a+b**

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)

11)

12) Output

13) The Results are

14) 150

15) 50

16) 5000

17) 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 are actual arguments are allowed in Python.

1. positional arguments
2. keyword arguments
3. default arguments
4. Variable length arguments
5. 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.

1. keyword arguments:

We can pass argument values by keyword i.e by parameter name. Eg:

|  |
| --- |
| 1. def wish(name,msg): |
| 2. print("Hello",name,msg) |
| 3. wish(name="Durga",msg="Good Morning") |
| 4. wish(msg="Good Morning",name="Durga") |
| 5. |
| 6. Output |
| 7. Hello Durga Good Morning |
| 8. 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.

def wish(name,msg):

print("Hello",name,msg) wish("Durga","GoodMorning") ==>valid wish("Durga",msg="GoodMorning") ==>valid wish(name="Durga","GoodMorning") ==>invalid SyntaxError: positional argument follows keyword argument

1. Default Arguments:

Sometimes we can provide default values for our positional arguments. Eg:

**1) def wish(name="Guest"): 3)**

**5) wish()**

**7) Output**

**9) Hello Guest Good Morning**

**8) Hello Durga Good Morning**

**6)**

**4) wish("Durga")**

**2) print("Hello",name,"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

def wish(name="Guest",msg="Good Morning"): ===>Valid def wish(name,msg="Good Morning"): ===>Valid

def wish(name="Guest",msg): ==>Invalid

SyntaxError: non-default argument follows default argument

1. 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.

Eg:

|  |
| --- |
| 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) |
| 11) |
| 12) Output |
| 13) The Sum= 0 |
| 14) The Sum= 10 |
| 15) The Sum= 30 |
| 16) The Sum= 100 |

Note:

We can mix variable length arguments with positional arguments.

Eg:

**1) def f1(n1,\*s):**

**3) for s1 in s: 5)**

**7) f1(10,20,30,40)**

**9)**

**11) 10**

**13) 20**

**15) 40**

**17) A**

**19) B**

**18) 30**

**16) 10**

**14) 30**

**12) 10**

**10) Output**

**8) f1(10,"A",30,"B")**

**6) f1(10)**

**print(s1)**

**4)**

**2) print(n1)**

Note: After variable length argument,if we are taking any other arguments then we should provide values as keyword arguments.

Eg:

|  |
| --- |
| 1) def f1(\*s,n1): |
| 2) for s1 in s: |
| 3) print(s1) |
| 4) print(n1) |
| 5) |
| 6) f1("A","B",n1=10) |
| 7) Output |
| 8) A |
| 9) B |
| 10) 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.

Eg:

|  |
| --- |
| 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") |
| 6) |
| 7) Output |
| 8) n1 = 10 |
| 9) n2 = 20 |
| 10) n3 = 30 |
| 11) rno = 100 |
| 12) name = Durga |
| 13) marks = 70 |
| 14) 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

1. 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

Library

Function

**Module 1 Module 2**

**Function 3**

**Function 2**

**Function 1**

**Function 3**

**Function 2**

**Function 1**

-----------------

-----------------

-----------------

-----------------

-----------------

-----------------

-----------------

-----------------

-----------------

Types of Variables

Python supports 2 types of variables.

1. Global Variables
2. Local Variables
3. 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.

Eg:

**1) a=10 # global variable**

**3) print(a)**

**5) def f2(): 7)**

**9) f2()**

**11) Output 13) 10**

**12) 10**

**10)**

**8) f1()**

**6) print(a)**

**4)**

**2) def f1():**

1. 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.

Eg:

**1) def f1():**

**3) print(a) # valid**

**5) def f2():**

**7)**

**9) f2()**

**11) NameError: name 'a' is not defined**

**10)**

**8) f1()**

**6) print(a) #invalid**

**4)**

**2) a=10**

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

Eg 1:

|  |
| --- |
| 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) |
| 12) Output |
| 13) 777 |
| 14) 10 |

Eg 2:

**1) a=10**

**3) global a**

**5) print(a)**

**7) def f2(): 9)**

**11) f2()**

**13) Output 15) 777**

**14) 777**

**12)**

**10) f1()**

**8) print(a)**

**6)**

**4) a=777**

**2) def f1():**

Eg 3:

**1) def f1():**

**3) print(a)**

**5) def f2():**

**7)**

**9) f2()**

**11) NameError: name 'a' is not defined**

**10)**

**8) f1()**

**6) print(a)**

**4)**

**2) a=10**

Eg 4:

|  |
| --- |
| 1) def f1(): |
| 2) global a |
| 3) a=10 |
| 4) print(a) |
| 5) |
| 6) def f2(): |
| 7) print(a) |
| 8) |
| 9) f1() |
| 10) f2() |
| 11) |
| 12) Output |
| 13) 10 |
| 14) 10 |

**12**

**DURGASOFT, # 202, 2 Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,**

**nd**

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**

**3) a=777 #local variable**

**5) print(globals()['a']) 7)**

**9) Output**

**11) 10**

**10) 777**

**8)**

**6) f1()**

**4) print(a)**

**2) def f1():**

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.

Eg:

|  |
| --- |
| 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)) |
| 9) |
| 10) Output |

**13**

**nd**

11) Factorial of 4 is : 24

12) 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**

**3) print("The Square of 5 is :",s(5))**

**5) Output**

**7) The Square of 5 is : 25**

**6) The Square of 4 is : 16**

**4)**

**2) print("The Square of 4 is :",s(4))**

Q. Lambda function to find sum of 2 given numbers

1. s=lambda a,b:a+b

**14**

**nd**

2) print("The Sum of 10,20 is:",s(10,20))

**3) print("The Sum of 100,200 is:",s(100,200))**

**5) Output**

**7) The Sum of 100,200 is: 300**

**6) The Sum of 10,20 is: 30**

**4)**

Q. Lambda Function to find biggest of given values.

**1) s=lambda a,b:a if a>b else b**

**3) print("The Biggest of 100,200 is:",s(100,200))**

**5) Output**

**7) The Biggest of 100,200 is: 200**

**6) The Biggest of 10,20 is: 20**

**4)**

**2) print("The Biggest of 10,20 is:",s(10,20))**

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,b'z 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:

**15**

**nd**

|  |
| --- |
| 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.

Eg: 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]

-------------------------------------------------------------

**16**

**nd**

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

Eg:

1. l1=[1,2,3,4]

2. l2=[2,3,4,5]

1. 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.

Eg:

**1) from functools import \* 2) l=[10,20,30,40,50]**

1. **result=reduce(lambda x,y:x+y,l)**
2. **print(result) # 150**

Eg:

**1) result=reduce(lambda x,y:x\*y,l)**

2) **print(result) #12000000**

Eg:

1. **from functools import \***
2. **result=reduce(lambda x,y:x+y,range(1,101))**
3. **print(result) #5050**

Note:

* In Python every thing is treated as object.
* Even functions also internally treated as objects only.

Eg:

|  |  |
| --- | --- |
| 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.

Eg:

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

Eg:

1) def wish(name):

2) print("Good Morning:",name)

**3)**

**5)**

**7) wish('Durga')**

**9) del wish**

**11) greeting('Pavan')**

**10) #wish('Durga') ==>NameError: name 'wish' is not defined**

**8)**

**6) greeting('Durga')**

**4) greeting=wish**

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.

Eg:

|  |  |
| --- | --- |
| 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.

Eg:

|  |  |
| --- | --- |
| 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 differenece 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)

**20**

**nd**

Function Decorators:

Decorator is a function which can take a function as argument and extend its functionality and returns modified function with extended functionality.

**Decorator**

Input Function wish()

new(add some functionality) inner()

Input Function

Output Function with extended Functionality

**21**

**nd**

The main objective of decorator functions is we can extend the functionality of existing functions without modifies that function.

**Decorator Function**

1. def wish(name):

2) print("Hello",name,"Good Morning")

This function can always print same output for any name

Hello Durga Good Morning Hello Ravi Good Morning Hello Sunny Good Morning

But we want to modify this function to provide different message if name is Sunny. We can do this without touching wish() function by using decorator.

Eg:

**1) def decor(func):**

**3)**

**if name=="Sunny":**

**5)**

**else:**

**7) return inner**

**9) @decor**

**11) print("Hello",name,"Good Morning")**

**13) wish("Durga")**

**15) wish("Sunny")**

**16)**

**14) wish("Ravi")**

**12)**

**10) def wish(name):**

**8)**

**func(name)**

**6)**

**print("Hello Sunny Bad Morning")**

**4)**

**2) def inner(name):**

|  |
| --- |
| 17) Output |
| 18) Hello Durga Good Morning |
| 19) Hello Ravi Good Morning |
| 20) Hello Sunny Bad Morning |

In the above program whenever we call wish() function automatically decor function will be executed.

How to call same function with decorator and without decorator:

We should not use @decor

**1) def decor(func):**

**3)**

**if name=="Sunny":**

**5)**

**else:**

**7) return inner**

**9)**

**11) print("Hello",name,"Good Morning")**

**13) decorfunction=decor(wish)**

**15) wish("Durga") #decorator wont be executed 17)**

**19) decorfunction("Sunny")#decorator will be executed**

**21) Output**

**23) Hello Sunny Good Morning**

**25) Hello Sunny Bad Morning**

**24) Hello Durga Good Morning**

**22) Hello Durga Good Morning**

**20)**

**18) decorfunction("Durga")#decorator will be executed**

**16) wish("Sunny") #decorator wont be executed**

**14)**

**12)**

**10) def wish(name):**

**8)**

**func(name)**

**6)**

**print("Hello Sunny Bad Morning")**

**4)**

**2) def inner(name):**

Eg 2:

|  |  |
| --- | --- |
| 1) | def smart\_division(func): |
| 2) | def inner(a,b): |
| 3) | print("We are dividing",a,"with",b) |
| 4) | if b==0: |
| 5) | print("OOPS...cannot divide") |
| 6) | return |
| 7) | else: |
| 8) | return func(a,b) |

|  |
| --- |
| 9) return inner |
| 10) |
| 11) @smart\_division |
| 12) def division(a,b): |
| 13) return a/b |
| 14) |
| 15) print(division(20,2)) |
| 16) print(division(20,0)) |
| 17) |
| 18) without decorator we will get Error.In this case output is: |
| 19) |
| 20) 10.0 |
| 21) Traceback (most recent call last): |
| 22) File "test.py", line 16, in <module> |
| 23) print(division(20,0)) |
| 24) File "test.py", line 13, in division |
| 25) return a/b |
| 26) ZeroDivisionError: division by zero |

with decorator we won't get any error. In this case output is:

We are dividing 20 with 2

10.0

We are dividing 20 with 0 OOPS...cannot divide None

Decorator Chaining

We can define multiple decorators for the same function and all these decorators will form Decorator Chaining.

Eg:

@decor1 @decor def num():

For num() function we are applying 2 decorator functions. First inner decorator will work and then outer decorator.

Eg:

|  |
| --- |
| 1) def decor1(func): |
| 2) def inner(): |
| 3) x=func() |
| 4) return x\*x |

**23**

**nd**

|  |
| --- |
| 5) return inner |
| 6) |
| 7) def decor(func): |
| 8) def inner(): |
| 9) x=func() |
| 10) return 2\*x |
| 11) return inner |
| 12) |
| 13) @decor1 |
| 14) @decor |

**24**

**nd**