**Python Programming**



##### 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**

**2) {**

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

**4) {**

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

**6) }**

**7) }**

**C:**

**1) #include<stdio.h>**

**2) void main()**

**3) {**

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

5) **}**

**Python:**

**print("Hello World")**



**Eg2: To print the sum of 2 numbers Java:**

|  |  |  |  |
| --- | --- | --- | --- |
| **1) public class Add** | | | |
| **2)** | **{** |  |  |
| **3) public static void main(String[] args)** | | | |
| **4)** |  | **{** |  |
| **5) int a,b;** | | | |
| **6)** |  |  | **a =10;** |
| **7)** |  |  | **b=20;** |
| **8) System.out.println("The Sum:"+(a+b));** | | | |
| **9)** |  | **}** |  |
| **10) }** | | | |

**C:**

**1) #include <stdio.h>**

**3) void main()**

**5) int a,b;**

**7) b=20;**

9) **}**

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

**6) a =10;**

**4) {**

**2)**

**Python:**

**1) a=10**

**2) b=20**

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

**The name Python was selected from the TV Show "The Complete**

**Monty Python's**

**Circus", which was broadcasted in BBC from 1969 to 1974.**

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

1. **Functional Programming Features from C**
2. **Object Oriented Programming Features from C++**
3. **Scripting Language Features from Perl and Shell Script**



1. **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.**



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. **Extensible:**

**We can use other language programs in Python. The main advantages of this approach are:**



1. **We can use already existing legacy non-Python code**
2. **We can improve performance of the application**
3. **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 represents the type of data present inside a variable.**

**In Python we are not required to specify the type explicitly. Based on value provided, the type will be assigned automatically.Hence Python is dynamically Typed Language.**

**Python contains the following inbuilt data types**

1. **int**
2. **float 3.complex 4.bool 5.str 6.bytes**

**7.bytearray 8.range 9.list 10.tuple 11.set 12.frozenset 13.dict 14.None**

**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()**

**to get address of object**



1. **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**

1. **Octal Form(Base-8):**

**The allowed digits are : 0 to 7**

**Literal value should be prefixed with 0o or 0O.**



**Eg: a=0o123**

**a=0o786**

1. **Hexa Decimal Form(Base-16):**

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

**Eg:**

**a =0XFACE**

**a=0XBeef a =0XBeer**

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

**a=10**

**b=0o10 c=0X10 d=0B10**

**print(a)10 print(b)8 print(c)16 print(d)2**

**Base Conversions**

**Python provide the following in-built functions for base conversions**

1. **bin():**

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

|  |  |
| --- | --- |
| **1)** | **>>> bin(15)** |
| **2)** | **'0b1111'** |
| **3)** | **>>> bin(0o11)** |
| **4)** | **'0b1001'** |
| **5)** | **>>> bin(0X10)** |
| **6)** | **'0b10000'** |

1. **oct():**

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



**Eg:**

|  |  |
| --- | --- |
| **1)** | **>>> oct(10)** |
| **2)** | **'0o12'** |
| **3)** | **>>> oct(0B1111)** |
| **4)** | **'0o17'** |
| **5)** | **>>> oct(0X123)** |
| 6) | **'0o443'** |

1. **hex():**

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

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

**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



|  |
| --- |
| **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**



**a + bj**

**j2 = -1**

**j =** √−1

**Real Part Imaginary Part**

**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**



**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**



|  |  |
| --- | --- |
| **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 various 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.**

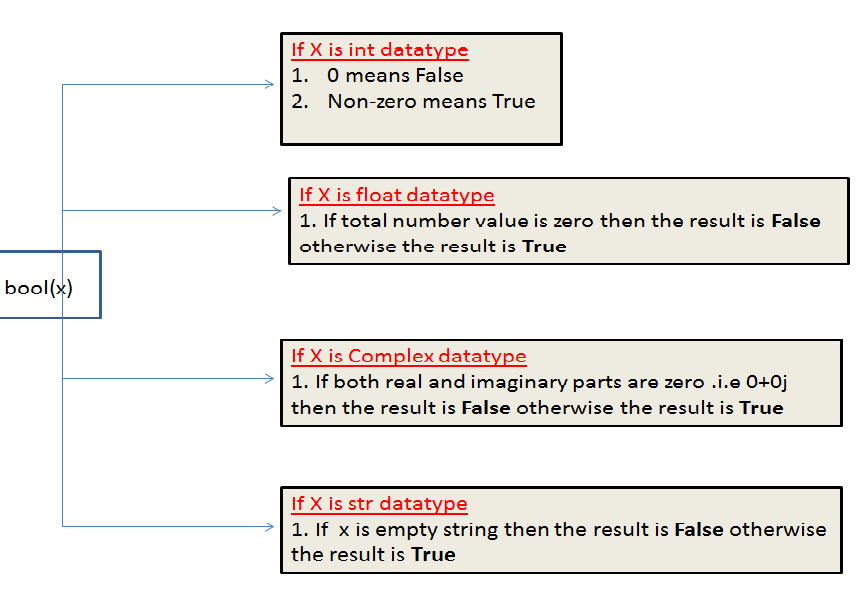
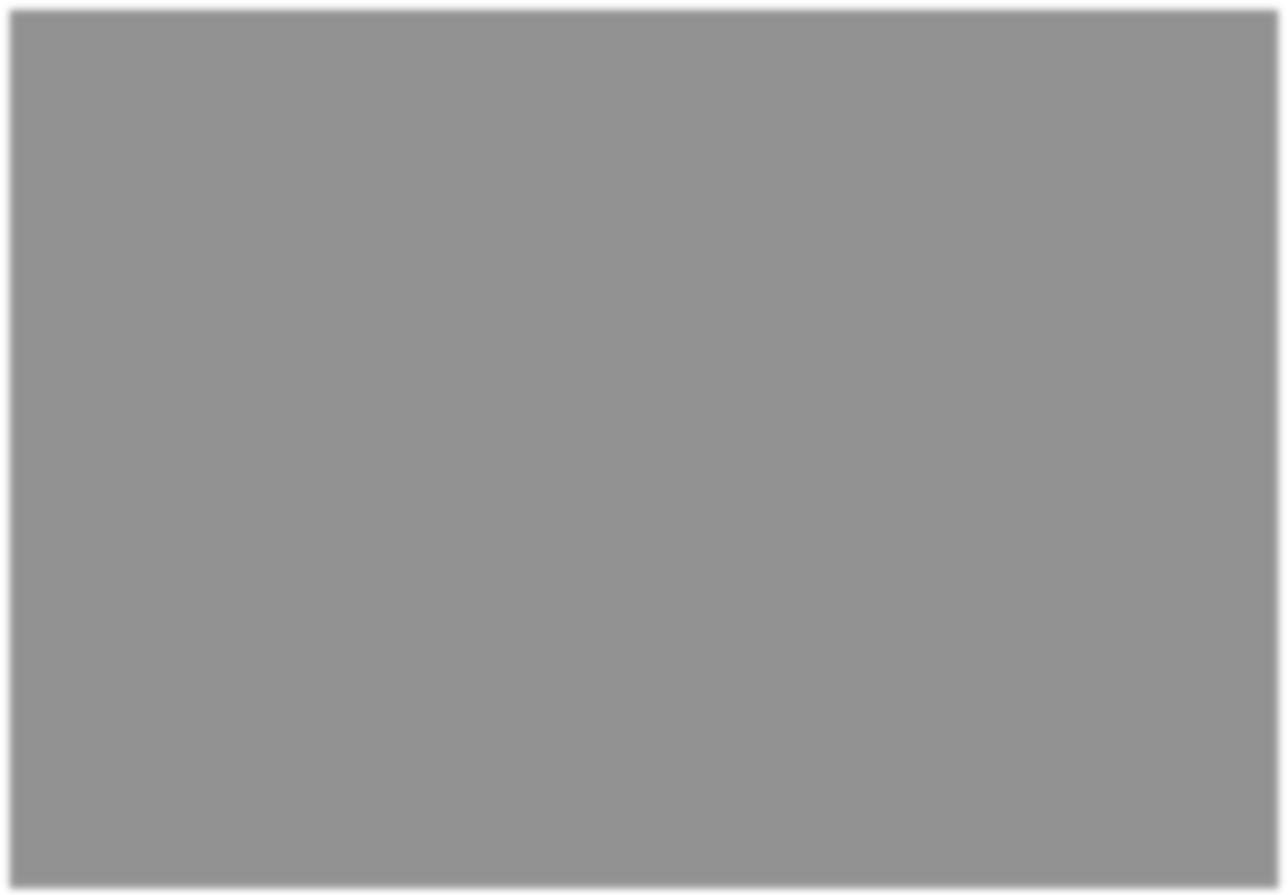
**Eg: complex(10,-2)==>10-2j complex(True,False)==>1+0j**

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]** |
| **2) print(list) # [10,10.5,'durga',True,10]** |

**Eg:**

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

2) >>> list[0]

3) 10

4) >>> list[-1]

5) 40

6) >>> list[1:3]

7) [20, 30]

8) >>> list[0]=100

9) >>> for i in list:print(i)

10) ...

11) 100

12) 20

13) 30



**14) 40**

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

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

**Note: An ordered, mutable, heterogenous collection of 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)** |



|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | **<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.**

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

2) print('h' in x) True

3) print('d' in x) False

4) print('d' not in x) True

5) print('Python' in x) True

**Eg:**

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

Operator Precedence:

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

**Eg:**

**print(3+10\*2)**  **23**

**print((3+10)\*2)**  **26**

**The following list describes operator precedence in Python ()**  **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)**

**2)**

**3)**

**4)**

**5)**

**6)**

**7)**

**8)**

**9)**

**a=30**

**b=20 c=10**

**d=5 print((a+b)\*c/d) print((a+b)\*(c/d)) print(a+(b\*c)/d)**

**100.0**

**100.0**

**70.0**

**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**



###### 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)** |
|  |



|  |
| --- |
| **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:")**

2) if name=="durga" :

3) print("Hello Durga Good Morning")

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

5)

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

7) Enter Name:durga

8) Hello Durga Good Morning

9) How are you!!!

10)

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

12) Enter Name:Ravi

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

* + 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) :** |
| **2) print(x)** |



**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):**

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

**3) continue**

**4) print(i)**

**5)**

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

**7) 1**

**8) 3**

**9) 5**

**10) 7**

11) **9**

**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():**

s.find(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"**
2. **l=s.split()**
3. **for x in l:**
4. **print(x)**

**Output: durga software solutions**

**Eg2:**

1) s="22-02-2018"

1. **l=s.split('-')**
2. **for x in l:**
3. **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**



|  |
| --- |
| **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:** |
| **2) def init (self,name,age):** |
| **3) self.name=name** |
| **4) self.age=age** |
| **5) print("{p.name}'s age is :{p.age}".format(p=Person('durga',48)))** |
| **6) print("{p.name}'s age is :{p.age}".format(p=Person('Ravi',50)))** |

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

Case-9: Dynamic Formatting using format()

1) string="{:{fill}{align}{width}}"

2) print(string.format('cat',fill='\*',align='^',width=5))

3) print(string.format('cat',fill='\*',align='^',width=6))

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

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



**Output:**

\*cat\*

\*cat\*\* cat\*\*\*

\*\*\*cat

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

1) num="{:{align}{width}.{precision}f}"

2) print(num.format(123.236,align='<',width=8,precision=2))

3) print(num.format(123.236,align='>',width=8,precision=2))

**Output:**

123.24

123.24

**Case-11: Formatting Date values**

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

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

**Case-12: Formatting complex numbers**

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

Output: Real Part:1.0 and Imaginary Part:2.0



**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=[]** |
| **2) print(list)** |
| **3) print(type(list))** |
| **4)** |
| **5) []** |
| **6) <class '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]** |
| **2) for n1 in n:** | |
| **3) print(n1)** | |
| **4)** | |
| **5) D:\Python\_classes>py test.py** | |
| **6)** | **0** |
| **7)** | **1** |
| **8)** | **2** |
| **9)** | **3** |
| **10) 4** | |
| **11) 5** | |
| **12) 6** | |
| **13) 7** | |
| **14) 8** | |
| **15) 9** | |
| **16) 10** | |

1. **To display only even numbers:**

|  |  |
| --- | --- |
| **1)** | **n=[0,1,2,3,4,5,6,7,8,9,10]** |
| **2) for n1 in n:** | |
| **3)** | **if n1%2==0:** |
| **4) print(n1)** | |
| **5)** | |
| **6) D:\Python\_classes>py test.py** | |
| **7)** | **0** |
| **8)** | **2** |
| **9)** | **4** |
| **10) 6** | |
| **11) 8** | |
| 12) | **10** |

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:**
2. **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. **order1=["Chicken","Mutton","Fish"]**

2) order2=["RC","KF","FO"]

3) order1.extend(order2)

4) print(order1)

5)

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

7) ['Chicken', 'Mutton', 'Fish', 'RC', 'KF', 'FO']

**Eg:**

|  |
| --- |
| **1) order=["Chicken","Mutton","Fish"]** |
| **2) order.extend("Mushroom")** |
| **3) print(order)** |
| **4)** |
| **5) D:\Python\_classes>py test.py** |
| **6) ['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=[]** |
| **2) print(n.pop()) ==> IndexError: pop from empty list** |



**Note:**

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

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

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

|  |
| --- |
| **1) n=[10,20,30,40,50,60]** |
| **2) print(n.pop()) #60** |
| **3) print(n.pop(1)) #20** |
| **4) print(n.pop(10)) ==>IndexError: pop index out of range** |

**Differences between remove() and pop()**

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

**Note:**

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

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



1. **Ordering elements of List:**
2. **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']** |
| **2) word=input("Enter the word to search for vowels: ")** |
| **3) found=[]** |
| **4) for letter in word:** |
| **5) if letter in vowels:** |
| **6) if letter not in found:** |
| **7) found.append(letter)** |
| **8) print(found)** |
| **9) print("The number of different vowels present in",word,"is",len(found))** |
| **10)** |
| **11)** |
| **12) D:\Python\_classes>py test.py** |



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)** |
| **3. 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

3. 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))**

2. print(type(t))

3. for x in t:

4. print(x)

5.

6. Output

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

8. <class 'generator'>

9. 1

10. 4

11. 9

12. 16

13. 25



**Q. Write a program to take a tuple of numbers from the keyboard and print its sum and average?**

**1. t=eval(input("Enter Tuple of Numbers:"))**

**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)

3. 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)** |
| **3. 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** |



**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** |



|  |
| --- |
| **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**

**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:**

|  |
| --- |
| **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]

**-------------------------------------------------------------**



**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]** |
| **3. l3=list(map(lambda x,y:x\*y,l1,l2))** |
| **4. print(l3) #[2, 6, 12, 20]** |

**reduce() function:**

**reduce() function reduces sequence of elements into a single element by applying the specified function.**

**reduce(function,sequence)**

**reduce() function present in functools module and hence we should write import statement.**

**Eg:**

|  |
| --- |
| **1) from functools import \*** |
| **2) l=[10,20,30,40,50]** |
| **3) result=reduce(lambda x,y:x+y,l)** |
| **4) print(result) # 150** |

**Eg:**

|  |
| --- |
| **1) result=reduce(lambda x,y:x\*y,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 functions 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)



**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**

**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):** |
| **2) def inner(name):** |
| **3) if name=="Sunny":** |
| **4) print("Hello Sunny Bad Morning")** |
| **5) else:** |
| **6) func(name)** |
| **7) return inner** |
| **8)** |
| **9) @decor** |
| **10) def wish(name):** |
| **11) print("Hello",name,"Good Morning")** |
| **12)** |
| **13) wish("Durga")** |
| **14) wish("Ravi")** |
| **15) wish("Sunny")** |
| **16)** |



|  |
| --- |
| **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** |



|  |
| --- |
| **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** |
| **15) def num():** |
| **16) return 10** |
| **17)** |
| **18) print(num())** |

Demo Program for decorator Chaining:

|  |
| --- |
| **1) def decor(func):** |
| **2) def inner(name):** |
| **3) print("First Decor(decor) Function Execution")** |
| **4) func(name)** |
| **5) return inner** |
| **6)** |
| **7) def decor1(func):** |
| **8) def inner(name):** |
| **9) print("Second Decor(decor1) Execution")** |
| **10) func(name)** |
| **11) return inner** |
| **12)** |
| **13) @decor1** |
| **14) @decor** |
| **15) def wish(name):** |
| **16) print("Hello",name,"Good Morning")** |
| **17)** |
| **18) wish("Durga")** |

**D:\durgaclasses>py decaratordemo1.py Second Decor(decor1) Execution**

**First Decor(decor) Function Execution Hello Durga Good Morning**



Generators

**Generator is a function which is responsible to generate a sequence of values.**

**We can write generator functions just like ordinary functions, but it uses yield keyword to return values.**

**A Sequence of Values**

**Generator Function**

**yield**

**Eg 1:**

|  |
| --- |
| **1) def mygen():** |
| **2) yield 'A'** |
| **3) yield 'B'** |
| **4) yield 'C'** |
| **5)** |
| **6) g=mygen()** |
| **7) print(type(g))** |
| **8)** |
| **9) print(next(g))** |
| **10) print(next(g))** |
| **11) print(next(g))** |
| **12) print(next(g))** |
| **13)** |
| **14) Output** |
| **15) <class 'generator'>** |
| **16) A** |
| **17) B** |
| **18) C** |
| **19) Traceback (most recent call last):** |
| **20) File "test.py", line 12, in <module>** |
| **21) print(next(g))** |
| **22) StopIteration** |

**Eg 2:**

|  |
| --- |
| **1) def countdown(num):** |
| **2) print("Start Countdown")** |
| **3) while(num>0):** |
| **4) yield num** |
| **5) num=num-1** |
| **6)** |
| **7) values=countdown(5)** |
| **8) for x in values:** |



9) print(x)

10)

11) Output

12) Start Countdown

13) 5

14) 4

15) 3

16) 2

17) 1

**Eg 3: To generate first n numbers:**

|  |
| --- |
| **1) def firstn(num):** |
| **2) n=1** |
| **3) while n<=num:** |
| **4) yield n** |
| **5) n=n+1** |
| **6)** |
| **7) values=firstn(5)** |
| **8) for x in values:** |
| **9) print(x)** |
| **10)** |
| **11) Output** |
| **12) 1** |
| **13) 2** |
| **14) 3** |
| **15) 4** |
| **16) 5** |

**We can convert generator into list as follows: values=firstn(10)**

**l1=list(values)**

**print(l1) #[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]**

**Eg 4: To generate Fibonacci Numbers... The next is the sum of previous 2 numbers Eg: 0,1,1,2,3,5,8,13,21,...**

**1) def fib():**

**3) while True:**

**5)**

**a,b=b,a+b**

**7) if f>100:**

**9) print(f)**

**break**

**8)**

**6) for f in fib():**

**yield a**

**4)**

**2) a,b=0,1**



|  |
| --- |
| **10)** |
| **11) Output** |
| **12) 0** |
| **13) 1** |
| **14) 1** |
| **15) 2** |
| **16) 3** |
| **17) 5** |
| **18) 8** |
| **19) 13** |
| **20) 21** |
| **21) 34** |
| **22) 55** |
| **23) 89** |

Advantages of Generator Functions:

1. **when compared with class level iterators, generators are very easy to use**
2. **Improves memory utilization and performance.**
3. **Generators are best suitable for reading data from large number of large files**
4. **Generators work great for web scraping and crawling.**

Generators vs Normal Collections wrt performance:

**1) import random**

**3)**

**5) subjects = ['Python','Java','Blockchain']**

**7) def people\_list(num\_people):**

**9) for i in range(num\_people):**

**11)**

**'id':i,**

**13)**

**'subject':random.choice(subjects)**

**15)**

**results.append(person)**

**17)**

**19) for i in range(num\_people):**

**21)**

**'id':i,**

**23)**

**'major':random.choice(subjects)**

**'name': random.choice(names),**

**22)**

**person = {**

**20)**

**18) def people\_generator(num\_people):**

**16) return results**

**}**

**14)**

**'name': random.choice(names),**

**12)**

**person = {**

**10)**

**8) results = []**

**6)**

**4) names = ['Sunny','Bunny','Chinny','Vinny']**

**2) import time**



|  |
| --- |
| **24) }** |
| **25) yield person** |
| **26)** |
| **27) '''''t1 = time.clock()** |
| **28) people = people\_list(10000000)** |
| **29) t2 = time.clock()'''** |
| **30)** |
| **31) t1 = time.clock()** |
| **32) people = people\_generator(10000000)** |
| **33) t2 = time.clock()** |
| **34)** |
| **35) print('Took {}'.format(t2-t1))** |

Note: In the above program observe the differnce wrt execution time by using list and generators

**Generators vs Normal Collections wrt Memory Utilization:**

**Normal Collection:**

l=[x\*x for x in range(10000000000000000)] print(l[0])

We will get MemoryError in this case because all these values are required to store in the memory.

**Generators:**

g=(x\*x for x in range(10000000000000000)) print(next(g))

Output: 0

We won't get any MemoryError because the values won't be stored at the beginning



# Modules

**A group of functions, variables and classes saved to a file, which is nothing but module. Every Python file (.py) acts as a module.**

**Eg: durgamath.py**

**1) x=888**

**3) def add(a,b):**

**5)**

7) **print("The Product:",a\*b)**

**6) def product(a,b):**

**4) print("The Sum:",a+b)**

**2)**

**durgamath module contains one variable and 2 functions.**

**If we want to use members of module in our program then we should import that module. import modulename**

**We can access members by using module name. modulename.variable**

**modulename.function() test.py:**

* 1. **import durgamath**

2) print(durgamath.x)

3) durgamath.add(10,20)

4) durgamath.product(10,20)

5)

6) Output

7) 888

8) The Sum: 30

9) The Product: 200



**Note:**

**whenever we are using a module in our program, for that module compiled file will be generated and stored in the hard disk permanently.**

**Renaming a module at the time of import (module aliasing):**

**Eg:**

**import durgamath as m**

**here durgamath is original module name and m is alias name.**

**We can access members by using alias name m test.py:**

|  |
| --- |
| **1) import durgamath as m** |
| **2) print(m.x)** |
| **3) m.add(10,20)** |
| **4) m.product(10,20)** |

**from ... import:**

**We can import particular members of module by using from ... import .**

**The main advantage of this is we can access members directly without using module name.**

**Eg:**

**from durgamath import x,add print(x)**

**add(10,20)**

**product(10,20)==> NameError: name 'product' is not defined**

**We can import all members of a module as follows from durgamath import \***

**test.py:**

|  |
| --- |
| **1) from durgamath import \*** |
| **2) print(x)** |
| **3) add(10,20)** |
| **4) product(10,20)** |



**Various possibilties of import:**

**import modulename**

**import module1,module2,module3 import module1 as m**

**import module1 as m1,module2 as m2,module3 from module import member**

**from module import member1,member2,memebr3 from module import memeber1 as x**

**from module import \***

**member aliasing:**

**from durgamath import x as y,add as sum print(y)**

**sum(10,20)**

**Once we defined as alias name,we should use alias name only and we should not use original name**

**Eg:**

**from durgamath import x as y print(x)==>NameError: name 'x' is not defined**

**Reloading a Module:**

**By default module will be loaded only once eventhough we are importing multiple multiple times.**

**Demo Program for module reloading:**

|  |
| --- |
| **1) import time** |
| **2) from imp import reload** |
| **3) import module1** |
| **4) time.sleep(30)** |
| **5) reload(module1)** |
| **6) time.sleep(30)** |
| **7) reload(module1)** |
| **8) print("This is test file")** |

**Note: In the above program, everytime updated version of module1 will be available to our program**



**module1.py:**

**print("This is from module1") test.py**

**1) import module1**

**3) import module1**

**5) print("This is test module")**

**7) Output**

**9) This is test module**

**8) This is from module1**

**6)**

**4) import module1**

**2) import module1**

**In the above program test module will be loaded only once eventhough we are importing multiple times.**

**The problem in this approach is after loading a module if it is updated outside then updated version of module1 is not available to our program.**

**We can solve this problem by reloading module explicitly based on our requirement. We can reload by using reload() function of imp module.**

**import imp imp.reload(module1)**

**test.py:**

1) import module1

2) import module1

3) from imp import reload

4) reload(module1)

5) reload(module1)

6) reload(module1)

7) print("This is test module")

**In the above program module1 will be loaded 4 times in that 1 time by default and 3 times explicitly. In this case output is**

1) This is from module1

2) This is from module1

3) This is from module1

4) This is from module1

5) This is test module



**The main advantage of explicit module reloading is we can ensure that updated version is always available to our program.**

**Finding members of module by using dir() function:**

**Python provides inbuilt function dir() to list out all members of current module or a specified module.**

**dir() ===>To list out all members of current module dir(moduleName)==>To list out all members of specified module**

**Eg 1: test.py**

|  |
| --- |
| **1) x=10** |
| **2) y=20** |
| **3) def f1():** |
| **4) print("Hello")** |
| **5) print(dir()) # To print all members of current module** |
| **6)** |
| **7) Output** |
| **8) [' annotations ', ' builtins ', ' cached ', ' doc ', ' file ', ' loader ', ' nam** |
| **e ', ' package ', ' spec ', 'f1', 'x', 'y']** |

**Eg 2: To display members of particular module: durgamath.py:**

**1) x=888**

**3) def add(a,b):**

**5)**

**7) print("The Product:",a\*b)**

**6) def product(a,b):**

**4) print("The Sum:",a+b)**

**2)**

**test.py:**

|  |
| --- |
| **1) import durgamath** |
| **2) print(dir(durgamath))** |
| **3)** |
| **4) Output** |
| **5) [' builtins ', ' cached ', ' doc ', ' file ', ' loader ', ' name ',** |
| **6) ' package ', ' spec ', 'add', 'product', 'x']** |

**Note: For every module at the time of execution Python interpreter will add some special properties automatically for internal use.**



**Eg: builtins , cached ,' doc , file , loader , name , package ,**

**spec**

**Based on our requirement we can access these properties also in our program. Eg: test.py:**

1) print( builtins )

2) print( cached )

3) print( doc )

4) print( file )

5) print( loader )

6) print( name )

7) print( package )

8) print( spec )

9)

10) Output

11) <module 'builtins' (built-in)>

12) None

13) None

**test.py**

|  |
| --- |
| **1) <\_frozen\_importlib\_external.SourceFileLoader object at 0x00572170>** |
| **2) main** |
| **3) None** |
| 4) **None** |

**The Special variable name :**

**For every Python program , a special variable name will be added internally. This variable stores information regarding whether the program is executed as an individual program or as a module.**

**If the program executed as an individual program then the value of this variable is**

**main**

**If the program executed as a module from some other program then the value of this variable is the name of module where it is defined.**

**Hence by using this name variable we can identify whether the program executed directly or as a module.**



**Demo program:**

**module1.py:**

|  |
| --- |
| **1) def f1():** |
| **2) if name ==' main ':** |
| **3) print("The code executed as a program")** |
| **4) else:** |
| **5) print("The code executed as a module from some other program")** |
| **6) f1()** |

**test.py:**

**1) import module1 3)**

**5) The code executed as a program**

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

**9) The code executed as a module from some other program**

**8) The code executed as a module from some other program**

**6)**

**4) D:\Python\_classes>py module1.py**

**2) module1.f1()**

**Working with math module:**

**Python provides inbuilt module math.**

**This module defines several functions which can be used for mathematical operations. The main important functions are**

1. **sqrt(x)**
2. **ceil(x)**
3. **floor(x)**
4. **fabs(x) 5.log(x)**
5. **sin(x)**
6. **tan(x)**

**....**

**Eg:**

|  |
| --- |
| **1) from math import \*** |
| **2) print(sqrt(4))** |
| **3) print(ceil(10.1))** |
| **4) print(floor(10.1))** |
| **5) print(fabs(-10.6))** |
| **6) print(fabs(10.6))** |



7)

8) Output

9) 2.0

10) 11

11) 10

12) 10.6

13) 10.6

**Note:**

**We can find help for any module by using help() function Eg:**

**import math help(math)**

**Working with random module:**

**This module defines several functions to generate random numbers.**

**We can use these functions while developing games,in cryptography and to generate random numbers on fly for authentication.**

1. **random() function:**

**This function always generate some float value between 0 and 1 ( not inclusive)**

**0<x<1**

**Eg:**

**1) from random import \***

**3)**

**print(random())**

**5) Output**

**7) 0.6584325233197768**

**9) 0.18351427005232201**

**11) 0.9291139798071045**

**13) 0.8901649834019002**

**15) 0.7290504335962871**

**14) 0.25540891083913053**

**12) 0.6586741197891783**

**10) 0.1330257265904884**

**8) 0.15444034016553587**

**6) 0.4572685609302056**

**4)**

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



1. **randint() function:**

**To generate random integer beween two given numbers(inclusive) Eg:**

**1) from random import \***

**3)**

**print(randint(1,100)) # generate random int value between 1 and 100(inclusive)**

**5) Output**

**7) 44**

**9) 70**

**11) 74**

**13) 10**

**15) 8**

**14) 40**

**12) 52**

**10) 49**

**8) 39**

**6) 51**

**4)**

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

1. **uniform():**

**It returns random float values between 2 given numbers(not inclusive) Eg:**

**1) from random import \***

**3)**

**print(uniform(1,10))**

**5) Output**

**7) 6.81102218793548**

**9) 8.567976357239834**

**11) 2.176137584071641**

**13) 6.0801725149678445**

**15) 1.9982221862917555**

**14) 7.508457735544763**

**12) 4.822867939432386**

**10) 6.363511674803802**

**8) 8.068672144377329**

**6) 9.787695398230332**

**4)**

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

**random() ===>in between 0 and 1 (not inclusive) randint(x,y) ==>in between x and y ( inclusive) uniform(x,y) ==> in between x and y ( not inclusive)**



1. **randrange([start],stop,[step])**

**returns a random number from range start<= x < stop**

**start argument is optional and default value is 0 step argument is optional and default value is 1**

**randrange(10)-->generates a number from 0 to 9 randrange(1,11)-->generates a number from 1 to 10 randrange(1,11,2)-->generates a number from 1,3,5,7,9**

**Eg 1:**

**1) from random import \***

**3)**

**print(randrange(10))**

**5) Output**

**7) 4**

**9) 2**

**11) 4**

**13) 9**

**15) 9**

**14) 5**

**12) 8**

**10) 9**

**8) 0**

**6) 9**

**4)**

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

**Eg 2:**

**1) from random import \***

**3)**

**print(randrange(1,11))**

**5) Output**

**7) 2**

**9) 10**

**11) 5**

**13) 1**

**15) 3**

**14) 6**

**12) 9**

**10) 3**

**8) 8**

**6) 2**

**4)**

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



**Eg 3:**

**1) from random import \***

**3)**

**print(randrange(1,11,2))**

**5) Output**

**7) 3**

**9) 5**

**11) 1**

**13) 1**

15) **3**

**14) 7**

**12) 1**

**10) 7**

**8) 9**

**6) 1**

**4)**

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

1. **choice() function:**

**It wont return random number.**

**It will return a random object from the given list or tuple. Eg:**

**1) from random import \***

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

**print(choice(list))**

**4)**

**2) list=["Sunny","Bunny","Chinny","Vinny","pinny"]**

Output Bunny pinny Bunny Sunny Bunny pinny pinny Vinny Bunny Sunny



# Packages

**It is an encapsulation mechanism to group related modules into a single unit.**

**package is nothing but folder or directory which represents collection of Python modules.**

**Any folder or directory contains init .py file,is considered as a Python package.This file can be empty.**

**A package can contains sub packages also.**

|  |  |  |
| --- | --- | --- |
| **x.py** |  | **y.py** |

**Loan**

**init .py**

**init .py**

**init .py**

**Home Loan**

**Vehicle Loan**

**n.py**

**m.py**

**File 1**

**File 1**

**File 1**



**init .py**

**init .py**

**init .py**

**Home Loan**

**Vehicle Loan**

**Module 1**

**Module 1**

**Module 1**

**Module 1**

**File 1**

**File 1**

**File 1**

**Loan**

**The main advantages of package statement are**

1. **We can resolve naming conflicts**
2. **We can identify our components uniquely**
3. **It improves modularity of the application Eg 1:**

**D:\Python\_classes>**

**|-test.py**

**|-pack1**

**|-module1.py**

**|- init .py**

**init .py:**

**empty file module1.py:**

**def f1():**

**print("Hello this is from module1 present in pack1")**

**test.py (version-1): import pack1.module1 pack1.module1.f1()**



**test.py (version-2):**

**from pack1.module1 import f1 f1()**

**Eg 2:**

**D:\Python\_classes>**

**|-test.py**

**|-com**

**|-module1.py**

**|- init .py**

**|-durgasoft**

**|-module2.py**

**|- init .py**

**init .py: empty file module1.py:**

**def f1():**

**print("Hello this is from module1 present in com") module2.py:**

**def f2():**

**print("Hello this is from module2 present in com.durgasoft") test.py:**

* 1. **from com.module1 import f1**
  2. **from com.durgasoft.module2 import f2 3. f1()**

4. f2() 5.

1. **Output**
2. **D:\Python\_classes>py test.py**
3. **Hello this is from module1 present in com**
4. **Hello this is from module2 present in com.durgasoft**

**Note: Summary diagram of library,packages,modules which contains functions,classes and variables.**



**Library**

**pack 1 pack 2 --------------- pack n**

**module 1 module 2 module n module 1 module 2 module n**

**function 1 function 2 function n variables classes**



**File Handling**

**As the part of programming requirement, we have to store our data permanently for future purpose. For this requirement we should go for files.**

**Files are very common permanent storage areas to store our data.**

**Types of Files:**

**There are 2 types of files**

1. **Text Files:**

**Usually we can use text files to store character data eg: abc.txt**

1. **Binary Files:**

**Usually we can use binary files to store binary data like images,video files, audio files etc...**

**Opening a File:**

**Before performing any operation (like read or write) on the file,first we have to open that file.For this we should use Python's inbuilt function open()**

**But at the time of open, we have to specify mode,which represents the purpose of opening file.**

**f = open(filename, mode)**

**The allowed modes in Python are**

1. **r**  **open an existing file for read operation. The file pointer is positioned at the beginning of the file.If the specified file does not exist then we will get FileNotFoundError.This is default mode.**



1. **w**  **open an existing file for write operation. If the file already contains some data then it will be overridden. If the specified file is not already avaialble then this mode will create that file.**
2. **a**  **open an existing file for append operation. It won't override existing data.If the specified file is not already avaialble then this mode will create a new file.**
3. **r+**  **To read and write data into the file. The previous data in the file will not be deleted.The file pointer is placed at the beginning of the file.**
4. **w+**  **To write and read data. It will override existing data.**
5. **a+**  **To append and read data from the file.It wont override existing data.**
6. **x**  **To open a file in exclusive creation mode for write operation. If the file already exists then we will get FileExistsError.**

**Note: All the above modes are applicable for text files. If the above modes suffixed with 'b' then these represents for binary files.**

**Eg: rb,wb,ab,r+b,w+b,a+b,xb**

**f = open("abc.txt","w")**

**We are opening abc.txt file for writing data.**

**Closing a File:**

**After completing our operations on the file,it is highly recommended to close the file. For this we have to use close() function.**

**f.close()**

**Various properties of File Object:**

**Once we opend a file and we got file object,we can get various details related to that file by using its properties.**

**name**  **Name of opened file**

**mode**  **Mode in which the file is opened**

**closed**  **Returns boolean value indicates that file is closed or not**

**readable()** **Retruns boolean value indicates that whether file is readable or not writable()** **Returns boolean value indicates that whether file is writable or not.**



**Eg:**

|  |
| --- |
| **1) f=open("abc.txt",'w')** |
| **2) print("File Name: ",f.name)** |
| **3) print("File Mode: ",f.mode)** |
| **4) print("Is File Readable: ",f.readable())** |
| **5) print("Is File Writable: ",f.writable())** |
| **6) print("Is File Closed : ",f.closed)** |
| **7) f.close()** |
| **8) print("Is File Closed : ",f.closed)** |
| **9)** |
| **10)** |
| **11) Output** |
| **12) D:\Python\_classes>py test.py** |
| **13) File Name: abc.txt** |
| **14) File Mode: w** |
| **15) Is File Readable: False** |
| **16) Is File Writable: True** |
| **17) Is File Closed : False** |
| **18) Is File Closed : True** |

**Writing data to text files:**

**We can write character data to the text files by using the following 2 methods.**

**write(str) writelines(list of lines)**

**Eg:**

|  |
| --- |
| **1) f=open("abcd.txt",'w')** |
| **2) f.write("Durga\n")** |
| **3) f.write("Software\n")** |
| **4) f.write("Solutions\n")** |
| **5) print("Data written to the file successfully")** |
| 6) **f.close()** |

**abcd.txt:**

**Durga Software Solutions**

**Note: In the above program, data present in the file will be overridden everytime if we run the program. Instead of overriding if we want append operation then we should open the file as follows.**

**f = open("abcd.txt","a")**



**Eg 2:**

* 1. **f=open("abcd.txt",'w')**

2) list=["sunny\n","bunny\n","vinny\n","chinny"]

3) f.writelines(list)

4) print("List of lines written to the file successfully")

5) **f.close()**

**abcd.txt:**

**sunny bunny vinny chinny**

**Note: while writing data by using write() methods, compulsory we have to provide line seperator(\n),otherwise total data should be written to a single line.**

**Reading Character Data from text files:**

**We can read character data from text file by using the following read methods.**

**read()** **To read total data from the file read(n)**  **To read 'n' characters from the file readline()** **To read only one line readlines()** **To read all lines into a list**

**Eg 1: To read total data from the file**

|  |
| --- |
| **1) f=open("abc.txt",'r')** |
| **2) data=f.read()** |
| **3) print(data)** |
| **4) f.close()** |
| **5)** |
| **6) Output** |
| **7) sunny** |
| **8) bunny** |
| **9) chinny** |
| **10) vinny** |

**Eg 2: To read only first 10 characters:**

1) f=open("abc.txt",'r')

2) data=f.read(10)

3) print(data)

4) f.close()

5)



6) Output

7) sunny

8) bunn

**Eg 3: To read data line by line:**

1) f=open("abc.txt",'r')

2) line1=f.readline()

3) print(line1,end='')

4) line2=f.readline()

5) print(line2,end='')

6) line3=f.readline()

7) print(line3,end='')

8) f.close()

9)

10) Output

11) sunny

12) bunny

13) chinny

**Eg 4: To read all lines into list:**

**1) f=open("abc.txt",'r')**

**3) for line in lines:**

**5) f.close()**

**7) Output**

**9) bunny**

**11) vinny**

**10) chinny**

**8) sunny**

**6)**

**4) print(line,end='')**

**2) lines=f.readlines()**

**Eg 5:**

1) f=open("abc.txt","r")

2) print(f.read(3))

3) print(f.readline())

4) print(f.read(4))

5) print("Remaining data")

6) print(f.read())

7)

8) Output

9) sun

10) ny

11)

12) bunn

13) Remaining data



14) y

15) chinny

16) vinny

**The with statement:**

**The with statement can be used while opening a file.We can use this to group file operation statements within a block.**

**The advantage of with statement is it will take care closing of file,after completing all operations automatically even in the case of exceptions also, and we are not required to close explicitly.**

**Eg:**

|  |
| --- |
| **1) with open("abc.txt","w") as f:** |
| **2) f.write("Durga\n")** |
| **3) f.write("Software\n")** |
| **4) f.write("Solutions\n")** |
| **5) print("Is File Closed: ",f.closed)** |
| **6) print("Is File Closed: ",f.closed)** |
| **7)** |
| **8) Output** |
| **9) Is File Closed: False** |
| **10) Is File Closed: True** |

**The seek() and tell() methods:**

**tell():**

**==>We can use tell() method to return current position of the cursor(file pointer) from beginning of the file. [ can you plese telll current cursor position]**

**The position(index) of first character in files is zero just like string index. Eg:**

|  |
| --- |
| **1) f=open("abc.txt","r")** |
| **2) print(f.tell())** |
| **3) print(f.read(2))** |
| **4) print(f.tell())** |
| **5) print(f.read(3))** |
| **6) print(f.tell())** |

**abc.txt: sunny bunny chinny vinny**



**Output:**

**0**

**su 2**

**nny 5**

**seek():**

**We can use seek() method to move cursor(file pointer) to specified location. [Can you please seek the cursor to a particular location]**

**f.seek(offset, fromwhere)**

**offset represents the number of positions**

**The allowed values for second attribute(from where) are 0 >From beginning of file(default value)**

1. **>From current position**
2. **>From end of the file**

**Note: Python 2 supports all 3 values but Python 3 supports only zero. Eg:**

* 1. **data="All Students are STUPIDS"**
  2. **f=open("abc.txt","w")**
  3. **f.write(data)**
  4. **with open("abc.txt","r+") as f:**
  5. **text=f.read()**
  6. **print(text)**
  7. **print("The Current Cursor Position: ",f.tell())**
  8. **f.seek(17)**
  9. **print("The Current Cursor Position: ",f.tell())**
  10. **f.write("GEMS!!!")**
  11. **f.seek(0)**
  12. **text=f.read()**
  13. **print("Data After Modification:")**
  14. **print(text) 15)**

16) Output 17)

1. **All Students are STUPIDS**
2. **The Current Cursor Position: 24**
3. **The Current Cursor Position: 17**
4. **Data After Modification:**



**22) All Students are GEMS!!!**

**How to check a particular file exists or not?**

**We can use os library to get information about files in our computer.**

**os module has path sub module,which contains isFile() function to check whether a particular file exists or not?**

**os.path.isfile(fname)**

**Q. Write a program to check whether the given file exists or not. If it is available then print its content?**

1) import os,sys

2) fname=input("Enter File Name: ")

3) if os.path.isfile(fname):

4) print("File exists:",fname)

5) f=open(fname,"r")

6) else:

7) print("File does not exist:",fname)

8) sys.exit(0)

9) print("The content of file is:")

10) data=f.read()

11) print(data)

12)

13) Output

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

15) Enter File Name: durga.txt

16) File does not exist: durga.txt

17)

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

19) Enter File Name: abc.txt

20) File exists: abc.txt

21) The content of file is:

22) All Students are GEMS!!!

23) All Students are GEMS!!!

24) All Students are GEMS!!!

25) All Students are GEMS!!!

26) All Students are GEMS!!!

27) All Students are GEMS!!!

**Note:**

**sys.exit(0) ===>To exit system without executing rest of the program.**

**argument represents status code . 0 means normal termination and it is the default value.**



**Q. Program to print the number of lines,words and characters present in the given file?**

1. **import os,sys**
2. **fname=input("Enter File Name: ")**
3. **if os.path.isfile(fname):**
4. **print("File exists:",fname)**
5. **f=open(fname,"r")**
6. **else:**
7. **print("File does not exist:",fname)**
8. **sys.exit(0)**
9. **lcount=wcount=ccount=0**
10. **for line in f:**
11. **lcount=lcount+1**
12. **ccount=ccount+len(line)**
13. **words=line.split()**
14. **wcount=wcount+len(words)**
15. **print("The number of Lines:",lcount)**
16. **print("The number of Words:",wcount)**
17. **print("The number of Characters:",ccount) 18)**
18. **Output**
19. **D:\Python\_classes>py test.py**
20. **Enter File Name: durga.txt**
21. **File does not exist: durga.txt 23)**
22. **D:\Python\_classes>py test.py**
23. **Enter File Name: abc.txt**
24. **File exists: abc.txt**
25. **The number of Lines: 6**
26. **The number of Words: 24**

29) The number of Characters: 149

**abc.txt:**

**All Students are GEMS!!! All Students are GEMS!!! All Students are GEMS!!! All Students are GEMS!!! All Students are GEMS!!! All Students are GEMS!!!**



**Handling Binary Data:**

**It is very common requirement to read or write binary data like images,video files,audio files etc.**

**Q. Program to read image file and write to a new image file?**

1) f1=open("rossum.jpg","rb")

2) f2=open("newpic.jpg","wb")

3) bytes=f1.read()

4) f2.write(bytes)

5) print("New Image is available with the name: newpic.jpg")

Handling csv files:

**CSV==>Comma seperated values**

**As the part of programming,it is very common requirement to write and read data wrt csv files. Python provides csv module to handle csv files.**

**Writing data to csv file:**

|  |
| --- |
| **1) import csv** |
| **2) with open("emp.csv","w",newline='') as f:** |
| **3) w=csv.writer(f) # returns csv writer object** |
| **4) w.writerow(["ENO","ENAME","ESAL","EADDR"])** |
| **5) n=int(input("Enter Number of Employees:"))** |
| **6) for i in range(n):** |
| **7) eno=input("Enter Employee No:")** |
| **8) ename=input("Enter Employee Name:")** |
| **9) esal=input("Enter Employee Salary:")** |
| **10) eaddr=input("Enter Employee Address:")** |
| **11) w.writerow([eno,ename,esal,eaddr])** |
| **12) print("Total Employees data written to csv file successfully")** |

**Note: Observe the difference with newline attribute and without with open("emp.csv","w",newline='') as f:**

**with open("emp.csv","w") as f:**

**Note: If we are not using newline attribute then in the csv file blank lines will be included between data. To prevent these blank lines, newline attribute is required in Python-3,but in Python-2 just we can specify mode as 'wb' and we are not required to use newline attribute.**



**Reading Data from csv file:**

|  |
| --- |
| **1) import csv** |
| **2) f=open("emp.csv",'r')** |
| **3) r=csv.reader(f) #returns csv reader object** |
| **4) data=list(r)** |
| **5) #print(data)** |
| **6) for line in data:** |
| **7) for word in line:** |
| **8) print(word,"\t",end='')** |
| **9) print()** |
| **10)** |
| **11) Output** |
| **12) D:\Python\_classes>py test.py** |
| **13) ENO ENAME ESAL EADDR** |
| **14) 100 Durga 1000 Hyd** |
| **15) 200 Sachin 2000 Mumbai** |
| **16) 300 Dhoni 3000 Ranchi** |

**Zipping and Unzipping Files:**

**It is very common requirement to zip and unzip files. The main advantages are:**

1. **To improve memory utilization**
2. **We can reduce transport time**
3. **We can improve performance.**

**To perform zip and unzip operations, Python contains one in-bulit module zip file. This module contains a class : ZipFile**

**To create Zip file:**

**We have to create ZipFile class object with name of the zip file,mode and constant ZIP\_DEFLATED. This constant represents we are creating zip file.**

**f = ZipFile("files.zip","w","ZIP\_DEFLATED")**

**Once we create ZipFile object,we can add files by using write() method.**

**f.write(filename)**



**Eg:**

1) from zipfile import \*

2) f=ZipFile("files.zip",'w',ZIP\_DEFLATED)

3) f.write("file1.txt")

4) f.write("file2.txt")

5) f.write("file3.txt")

6) f.close()

7) print("files.zip file created successfully")

**To perform unzip operation:**

**We have to create ZipFile object as follows**

**f = ZipFile("files.zip","r",ZIP\_STORED)**

**ZIP\_STORED represents unzip operation. This is default value and hence we are not required to specify.**

**Once we created ZipFile object for unzip operation,we can get all file names present in that zip file by using namelist() method.**

**names = f.namelist()**

**Eg:**

1. **from zipfile import \***
2. **f=ZipFile("files.zip",'r',ZIP\_STORED)**
3. **names=f.namelist()**
4. **for name in names:**
5. **print( "File Name: ",name)**
6. **print("The Content of this file is:")**
7. **f1=open(name,'r')**
8. **print(f1.read())**

9) **print()**

**Working with Directories:**

**It is very common requirement to perform operations for directories like**

1. **To know current working directory**
2. **To create a new directory**
3. **To remove an existing directory**
4. **To rename a directory**
5. **To list contents of the directory etc...**



**To perform these operations,Python provides inbuilt module os,which contains several functions to perform directory related operations.**

**Q1. To Know Current Working Directory:**

**import os cwd=os.getcwd()**

**print("Current Working Directory:",cwd)**

**Q2. To create a sub directory in the current working directory:**

**import os os.mkdir("mysub")**

**print("mysub directory created in cwd")**

**Q3. To create a sub directory in mysub directory:**

**cwd**

**|-mysub**

**|-mysub2**

**import os os.mkdir("mysub/mysub2")**

**print("mysub2 created inside mysub")**

**Note: Assume mysub already present in cwd.**

**Q4. To create multiple directories like sub1 in that sub2 in that sub3:**

**import os os.makedirs("sub1/sub2/sub3")**

**print("sub1 and in that sub2 and in that sub3 directories created")**

**Q5. To remove a directory:**

**import os os.rmdir("mysub/mysub2") print("mysub2 directory deleted")**

**Q6. To remove multiple directories in the path:**

**import os os.removedirs("sub1/sub2/sub3")**

**print("All 3 directories sub1,sub2 and sub3 removed")**



**Q7. To rename a directory:**

**import os os.rename("mysub","newdir")**

**print("mysub directory renamed to newdir")**

**Q8. To know contents of directory:**

**os module provides listdir() to list out the contents of the specified directory. It won't display the contents of sub directory.**

**Eg:**

* 1. **import os**

2) print(os.listdir("."))

3)

4) Output

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

6) ['abc.py', 'abc.txt', 'abcd.txt', 'com', 'demo.py', 'durgamath.py', 'emp.csv', '

7) file1.txt', 'file2.txt', 'file3.txt', 'files.zip', 'log.txt', 'module1.py', 'myl

8) og.txt', 'newdir', 'newpic.jpg', 'pack1', 'rossum.jpg', 'test.py', ' pycache '

9) **]**

**The above program display contents of current working directory but not contents of sub directories.**

**If we want the contents of a directory including sub directories then we should go for walk() function.**

**Q9. To know contents of directory including sub directories:**

**We have to use walk() function**

**[Can you please walk in the directory so that we can aware all contents of that directory] os.walk(path,topdown=True,onerror=None,followlinks=False)**

**It returns an Iterator object whose contents can be displayed by using for loop**

**path-->Directory path. cwd means . topdown=True --->Travel from top to bottom**

**onerror=None --->on error detected which function has to execute. followlinks=True -->To visit directories pointed by symbolic links**



**Eg: To display all contents of Current working directory including sub directories:**

|  |
| --- |
| **1) import os** |
| **2) for dirpath,dirnames,filenames in os.walk('.'):** |
| **3) print("Current Directory Path:",dirpath)** |
| **4) print("Directories:",dirnames)** |
| **5) print("Files:",filenames)** |
| **6) print()** |
| **7)** |
| **8)** |
| **9) Output** |
| **10) Current Directory Path: .** |
| **11) Directories: ['com', 'newdir', 'pack1', ' pycache ']** |
| **12) Files: ['abc.txt', 'abcd.txt', 'demo.py', 'durgamath.py', 'emp.csv', 'file1.txt'** |
| **13) , 'file2.txt', 'file3.txt', 'files.zip', 'log.txt', 'module1.py', 'mylog.txt', '** |
| **14) newpic.jpg', 'rossum.jpg', 'test.py']** |
| **15)** |
| **16) Current Directory Path: .\com** |
| **17) Directories: ['durgasoft', ' pycache ']** |
| **18) Files: ['module1.py', ' init .py']** |
| **19)** |
| **20) ...** |

**Note: To display contents of particular directory,we have to provide that directory name as argument to walk() function.**

**os.walk("directoryname")**

**Q. What is the difference between listdir() and walk() functions?**

**In the case of listdir(), we will get contents of specified directory but not sub directory contents. But in the case of walk() function we will get contents of specified directory and its sub directories also.**

**Running Other programs from Python program:**

**os module contains system() function to run programs and commands. It is exactly same as system() function in C language.**

**os.system("commad string")**

**The argument is any command which is executing from DOS.**

**Eg:**

**import os os.system("dir \*.py") os.system("py abc.py")**



**How to get information about a File:**

**We can get statistics of a file like size, last accessed time,last modified time etc by using stat() function of os module.**

**stats = os.stat("abc.txt")**

**The statistics of a file includes the following parameters: st\_mode==>Protection Bits**

**st\_ino==>Inode number st\_dev===>device st\_nlink===>no of hard links st\_uid===>userid of owner st\_gid==>group id of owner st\_size===>size of file in bytes**

**st\_atime==>Time of most recent access st\_mtime==>Time of Most recent modification st\_ctime==> Time of Most recent meta data change**

**Note:**

**st\_atime, st\_mtime and st\_ctime returns the time as number of milli seconds since Jan 1st 1970 ,12:00AM. By using datetime module fromtimestamp() function,we can get exact date and time.**

**Q. To print all statistics of file abc.txt:**

1. **import os**
2. **stats=os.stat("abc.txt")**
3. **print(stats)**

4)

5) Output

6) os.stat\_result(st\_mode=33206, st\_ino=844424930132788, st\_dev=2657980798, st\_nlin 7) k=1, st\_uid=0, st\_gid=0, st\_size=22410, st\_atime=1505451446, st\_mtime=1505538999

8) , st\_ctime=1505451446)

**Q. To print specified properties:**

1. **import os**
2. **from datetime import \***
3. **stats=os.stat("abc.txt")**
4. **print("File Size in Bytes:",stats.st\_size)**
5. **print("File Last Accessed Time:",datetime.fromtimestamp(stats.st\_atime))**
6. **print("File Last Modified Time:",datetime.fromtimestamp(stats.st\_mtime)) 7)**



|  |
| --- |
| **8) Output** |
| **9) File Size in Bytes: 22410** |
| **10) File Last Accessed Time: 2017-09-15 10:27:26.599490** |
| **11) File Last Modified Time: 2017-09-16 10:46:39.245394** |

Pickling and Unpickling of Objects:

**Sometimes we have to write total state of object to the file and we have to read total object from the file.**

**The process of writing state of object to the file is called pickling and the process of reading state of an object from the file is called unpickling.**

**We can implement pickling and unpickling by using pickle module of Python. pickle module contains dump() function to perform pickling.**

**pickle.dump(object,file)**

**pickle module contains load() function to perform unpickling**

**obj=pickle.load(file)**

pickling

e1

**eno: 100**

**ename: Durga esal: 10000 eaddr: HYD**

**pickle.dump**

**(e1, f)**

**eno: 100**

**ename: Durga esal: 10000 eaddr: HYD**

**eno: 100**

**ename: Durga esal: 10000 eaddr: HYD**

**abc.txt**

**pickle load (f)**

**unpickling**

e2



**Writing and Reading State of object by using pickle Module:**

|  |
| --- |
| **1) import pickle** |
| **2) class Employee:** |
| **3) def init (self,eno,ename,esal,eaddr):** |
| **4) self.eno=eno;** |
| **5) self.ename=ename;** |
| **6) self.esal=esal;** |
| **7) self.eaddr=eaddr;** |
| **8) def display(self):** |
| **9) print(self.eno,"\t",self.ename,"\t",self.esal,"\t",self.eaddr)** |
| **10) with open("emp.dat","wb") as f:** |
| **11) e=Employee(100,"Durga",1000,"Hyd")** |
| **12) pickle.dump(e,f)** |
| **13) print("Pickling of Employee Object completed...")** |
| 14) |
| **15) with open("emp.dat","rb") as f:** |
| **16) obj=pickle.load(f)** |
| **17) print("Printing Employee Information after unpickling")** |
| **18) obj.display()** |

**Writing Multiple Employee Objects to the file:**

**emp.py:**

1) class Employee:

|  |
| --- |
| **2) def init (self,eno,ename,esal,eaddr):** |
| **3) self.eno=eno;** |
| **4) self.ename=ename;** |
| **5) self.esal=esal;** |
| 1. **self.eaddr=eaddr;** 2. **def display(self):** |
| **8)** |
| **9) print(self.eno,"\t",self.ename,"\t",self.esal,"\t",self.eaddr)** |

**pick.py:**

1) import emp,pickle

2) f=open("emp.dat","wb")

3) n=int(input("Enter The number of Employees:"))

4) for i in range(n):

5) eno=int(input("Enter Employee Number:"))

6) ename=input("Enter Employee Name:")

7) esal=float(input("Enter Employee Salary:"))

8) eaddr=input("Enter Employee Address:")

9) e=emp.Employee(eno,ename,esal,eaddr)

10) pickle.dump(e,f)

11) print("Employee Objects pickled successfully")



**19**

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

**nd**

**unpick.py:**

**1) import emp,pickle**

**3) print("Employee Details:")**

**5) try:**

**7)**

**obj.display()**

**9)**

**print("All employees Completed")**

**11) f.close()**

**break**

**10)**

**8) except EOFError:**

**obj=pickle.load(f)**

**6)**

**4) while True:**

**2) f=open("emp.dat","rb")**



**Exception Handling**

**In any programming language there are 2 types of errors are possible.**

1. **Syntax Errors**
2. **Runtime Errors**
3. **Syntax Errors:**

**The errors which occurs because of invalid syntax are called syntax errors. Eg 1:**

**x=10**

**if x==10 print("Hello")**

**SyntaxError: invalid syntax Eg 2:**

**print "Hello"**

**SyntaxError: Missing parentheses in call to 'print'**

**Note:**

**Programmer is responsible to correct these syntax errors. Once all syntax errors are corrected then only program execution will be started.**

1. **Runtime Errors:**

**Also known as exceptions.**

**While executing the program if something goes wrong because of end user input or programming logic or memory problems etc then we will get Runtime Errors.**

**Eg: print(10/0) ==>ZeroDivisionError: division by zero**

**print(10/"ten") ==>TypeError: unsupported operand type(s) for /: 'int' and 'str' x=int(input("Enter Number:"))**

**print(x) D:\Python\_classes>py test.py**



**Enter Number:ten**

**ValueError: invalid literal for int() with base 10: 'ten'**

**Note: Exception Handling concept applicable for Runtime Errors but not for syntax errors**

**What is Exception:**

**An unwanted and unexpected event that disturbs normal flow of program is called exception.**

**Eg:**

**ZeroDivisionError TypeError ValueError FileNotFoundError EOFError SleepingError TyrePuncturedError**

**It is highly recommended to handle exceptions. The main objective of exception handling is Graceful Termination of the program(i.e we should not block our resources and we should not miss anything)**

**Exception handling does not mean repairing exception. We have to define alternative way to continue rest of the program normally.**

**Eg:**

**For example our programming requirement is reading data from remote file locating at London. At runtime if london file is not available then the program should not be terminated abnormally. We have to provide local file to continue rest of the program normally. This way of defining alternative is nothing but exception handling.**

**try:**

**read data from remote file locating at london except FileNotFoundError:**

**use local file and continue rest of the program normally**

**Q. What is an Exception?**

**Q. What is the purpose of Exception Handling?**

**Q. What is the meaning of Exception Handling?**



**Default Exception Handing in Python:**

**Every exception in Python is an object. For every exception type the corresponding classes are available.**

**Whevever an exception occurs PVM will create the corresponding exception object and will check for handling code. If handling code is not available then Python interpreter terminates the program abnormally and prints corresponding exception information to the console.**

**The rest of the program won't be executed. Eg:**

|  |
| --- |
| **1) print("Hello")** |
| **2) print(10/0)** |
| **3) print("Hi")** |
| **4)** |
| **5) D:\Python\_classes>py test.py** |
| **6) Hello** |
| **7) Traceback (most recent call last):** |
| **8) File "test.py", line 2, in <module>** |
| **9) print(10/0)** |
| **10) ZeroDivisionError: division by zero** |



**Python's Exception Hierarchy**

**KeyboardInterrupt**

**GeneratorExit**

**SystemExit**

**Exception**

**BaseException**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Attribute** |  | **Arithmetic** |  | **EOF** |  | **Name** |  | **Lookup** |  | **OS** |  | **Type** |  | **Value** |
| **Error** |  | **Error** |  | **Error** |  | **Error** |  | **Error** |  | **Error** |  | **Error** |  | **Error** |

**Every Exception in Python is a class.**

**Index**

**Error**

**FileNotFound**

**Error**

**Key**

**Error**

**Interrupted**

**Error**

**Permission**

**Error**

**TimeOut Error**

**ZeroDivision**

**Error**

**FloatingPoint**

**Error**

**Overflow**

**Error**

**All exception classes are child classes of BaseException.i.e every exception class extends BaseException either directly or indirectly. Hence BaseException acts as root for Python Exception Hierarchy.**

**Most of the times being a programmer we have to concentrate Exception and its child classes.**

**Customized Exception Handling by using try-except:**

**It is highly recommended to handle exceptions.**

**The code which may raise exception is called risky code and we have to take risky code inside try block. The corresponding handling code we have to take inside except block.**



**try:**

**Risky Code except XXX:**

**Handling code/Alternative Code**

**without try-except:**

**1. print("stmt-1")**

**3. print("stmt-3")**

**5. Output**

**7. ZeroDivisionError: division by zero**

**6. stmt-1**

**4.**

**2. print(10/0)**

**Abnormal termination/Non-Graceful Termination**

**with try-except:**

1. print("stmt-1")

2. try:

3. print(10/0)

4. except ZeroDivisionError:

5. print(10/2)

6. print("stmt-3")

7.

8. Output

9. stmt-1

10. 5.0

11. stmt-3

**Normal termination/Graceful Termination**

**Control Flow in try-except:**

**try:**

**stmt-1 stmt-2 stmt-3**

**except XXX:**

**stmt-4 stmt-5**

**case-1: If there is no exception 1,2,3,5 and Normal Termination**



**case-2: If an exception raised at stmt-2 and corresponding except block matched 1,4,5 Normal Termination**

**case-3: If an exception raised at stmt-2 and corresponding except block not matched 1, Abnormal Termination**

**case-4: If an exception raised at stmt-4 or at stmt-5 then it is always abnormal termination.**

**Conclusions:**

1. **within the try block if anywhere exception raised then rest of the try block wont be executed eventhough we handled that exception. Hence we have to take only risky code inside try block and length of the try block should be as less as possible.**
2. **In addition to try block,there may be a chance of raising exceptions inside except and finally blocks also.**
3. **If any statement which is not part of try block raises an exception then it is always abnormal termination.**

**How to print exception information:**

**try:**

**1. print(10/0)**

**3. print("exception raised and its description is:",msg)**

**5. Output exception raised and its description is: division by zero**

**4.**

**2. except ZeroDivisionError as msg:**

**try with multiple except blocks:**

**The way of handling exception is varied from exception to exception.Hence for every exception type a seperate except block we have to provide. i.e try with multiple except blocks is possible and recommended to use.**

**Eg:**

**try:**

**-------**

**-------**

**-------**

**except ZeroDivisionError: perform alternative arithmetic operations**



**except FileNotFoundError:**

**use local file instead of remote file**

**If try with multiple except blocks available then based on raised exception the corresponding except block will be executed.**

**Eg:**

* 1. **try:**

2) x=int(input("Enter First Number: "))

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

4) print(x/y)

5) except ZeroDivisionError :

6) print("Can't Divide with Zero")

7) except ValueError:

8) print("please provide int value only")

9)

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

11) Enter First Number: 10

12) Enter Second Number: 2

13) 5.0

14)

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

16) Enter First Number: 10

17) Enter Second Number: 0

18) Can't Divide with Zero

19)

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

21) Enter First Number: 10

22) Enter Second Number: ten

23) please provide int value only

**If try with multiple except blocks available then the order of these except blocks is important .Python interpreter will always consider from top to bottom until matched except block identified.**

**Eg:**

|  |
| --- |
| **1) try:** |
| **2) x=int(input("Enter First Number: "))** |
| **3) y=int(input("Enter Second Number: "))** |
| **4) print(x/y)** |
| **5) except ArithmeticError :** |
| **6) print("ArithmeticError")** |
| **7) except ZeroDivisionError:** |
| **8) print("ZeroDivisionError")** |
| **9)** |
| **10) D:\Python\_classes>py test.py** |



11) Enter First Number: 10

12) Enter Second Number: 0

13) ArithmeticError

**Single except block that can handle multiple exceptions:**

**We can write a single except block that can handle multiple different types of exceptions.**

**except (Exception1,Exception2,exception3,..): or except (Exception1,Exception2,exception3,..) as msg :**

**Parenthesis are mandatory and this group of exceptions internally considered as tuple. Eg:**

1) try:

2) x=int(input("Enter First Number: "))

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

4) print(x/y)

5) except (ZeroDivisionError,ValueError) as msg:

6) print("Plz Provide valid numbers only and problem is: ",msg)

7)

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

9) Enter First Number: 10

10) Enter Second Number: 0

11) Plz Provide valid numbers only and problem is: division by zero

12)

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

14) Enter First Number: 10

15) Enter Second Number: ten

16) Plz Provide valid numbers only and problem is: invalid literal for int() with b

17) ase 10: 'ten'

**Default except block:**

**We can use default except block to handle any type of exceptions.**

**In default except block generally we can print normal error messages.**

**Syntax: except:**

**statements**

**Eg:**

|  |
| --- |
| **1) try:** |
| **2) x=int(input("Enter First Number: "))** |
| **3) y=int(input("Enter Second Number: "))** |
| **4) print(x/y)** |



|  |
| --- |
| **5) except ZeroDivisionError:** |
| **6) print("ZeroDivisionError:Can't divide with zero")** |
| **7) except:** |
| **8) print("Default Except:Plz provide valid input only")** |
| **9)** |
| **10) D:\Python\_classes>py test.py** |
| **11) Enter First Number: 10** |
| **12) Enter Second Number: 0** |
| **13) ZeroDivisionError:Can't divide with zero** |
| **14)** |
| **15) D:\Python\_classes>py test.py** |
| **16) Enter First Number: 10** |
| **17) Enter Second Number: ten** |
| **18) Default Except:Plz provide valid input only** |

**\*\*\*Note: If try with multiple except blocks available then default except block should be last,otherwise we will get SyntaxError.**

**Eg:**

|  |
| --- |
| **1) try:** |
| **2) print(10/0)** |
| **3) except:** |
| **4) print("Default Except")** |
| **5) except ZeroDivisionError:** |
| **6) print("ZeroDivisionError")** |
| **7)** |
| **8) SyntaxError: default 'except:' must be last** |

**Note:**

**The following are various possible combinations of except blocks**

**1. except ZeroDivisionError:**

**1. except ZeroDivisionError as msg:**

1. **except (ZeroDivisionError,ValueError) :**
2. **except (ZeroDivisionError,ValueError) as msg:**
3. **except :**

**finally block:**

1. **It is not recommended to maintain clean up code(Resource Deallocating Code or Resource Releasing code) inside try block because there is no guarentee for the execution of every statement inside try block always.**
2. **It is not recommended to maintain clean up code inside except block, because if there is no exception then except block won't be executed.**



**Hence we required some place to maintain clean up code which should be executed always irrespective of whether exception raised or not raised and whether exception handled or not handled. Such type of best place is nothing but finally block.**

**Hence the main purpose of finally block is to maintain clean up code.**

**try:**

**Risky Code**

**except:**

**Handling Code finally:**

**Cleanup code**

**The speciality of finally block is it will be executed always whether exception raised or not raised and whether exception handled or not handled.**

**Case-1: If there is no exception**

|  |
| --- |
| **1) try:** |
| **2) print("try")** |
| **3) except:** |
| **4) print("except")** |
| **5) finally:** |
| **6) print("finally")** |
| **7)** |
| **8) Output** |
| **9) try** |
| **10) finally** |

**Case-2: If there is an exception raised but handled:**

|  |
| --- |
| **1) try:** |
| **2) print("try")** |
| **3) print(10/0)** |
| **4) except ZeroDivisionError:** |
| **5) print("except")** |
| **6) finally:** |
| **7) print("finally")** |
| **8)** |
| **9) Output** |
| **10) try** |
| **11) except** |
| **12) finally** |



**Case-3: If there is an exception raised but not handled:**

|  |
| --- |
| **1) try:** |
| **2) print("try")** |
| **3) print(10/0)** |
| **4) except NameError:** |
| **5) print("except")** |
| **6) finally:** |
| **7) print("finally")** |
| **8)** |
| **9) Output** |
| **10) try** |
| **11) finally** |
| **12) ZeroDivisionError: division by zero(Abnormal Termination)** |

**\*\*\* Note: There is only one situation where finally block won't be executed ie whenever we are using os.\_exit(0) function.**

**Whenever we are using os.\_exit(0) function then Python Virtual Machine itself will be shutdown.In this particular case finally won't be executed.**

* 1. **imports**

2) try:

3) print("try")

4) os.\_exit(0)

5) except NameError:

6) print("except")

7) finally:

8) print("finally")

9)

10) Output

11) try

**Note:**

**os.\_exit(0)**

**where 0 represents status code and it indicates normal termination There are multiple status codes are possible.**

**Control flow in try-except-finally:**

**try:**

**stmt-1 stmt-2 stmt-3**

**except: stmt-4**



**finally:**

**stmt-5 stmt6**

**Case-1: If there is no exception 1,2,3,5,6 Normal Termination**

**Case-2: If an exception raised at stmt2 and the corresponding except block matched 1,4,5,6 Normal Termination**

**Case-3: If an exception raised at stmt2 but the corresponding except block not matched 1,5 Abnormal Termination**

**Case-4:If an exception raised at stmt4 then it is always abnormal termination but before that finally block will be executed.**

**Case-5: If an exception raised at stmt-5 or at stmt-6 then it is always abnormal termination**

**Nested try-except-finally blocks:**

**We can take try-except-finally blocks inside try or except or finally blocks.i.e nesting of try- except-finally is possible.**

**try:**

**----------**

**----------**

**----------**

**try:**

**-------------**

**--------------**

**--------------**

**except:**

**--------------**

**--------------**

**--------------**

**------------**

**except:**

**-----------**

**-----------**

**-----------**

**General Risky code we have to take inside outer try block and too much risky code we have to take inside inner try block. Inside Inner try block if an exception raised then inner**



**except block is responsible to handle. If it is unable to handle then outer except block is responsible to handle.**

**Eg:**

|  |
| --- |
| **1) try:** |
| **2) print("outer try block")** |
| **3) try:** |
| **4) print("Inner try block")** |
| **5) print(10/0)** |
| **6) except ZeroDivisionError:** |
| **7) print("Inner except block")** |
| **8) finally:** |
| **9) print("Inner finally block")** |
| **10) except:** |
| **11) print("outer except block")** |
| **12) finally:** |
| **13) print("outer finally block")** |
| **14)** |
| **15) Output** |
| **16) outer try block** |
| **17) Inner try block** |
| **18) Inner except block** |
| **19) Inner finally block** |
| **20) outer finally block** |

**Control flow in nested try-except-finally:**

**try:**

**stmt-1 stmt-2 stmt-3 try:**

**stmt-4 stmt-5 stmt-6**

**except X:**

**stmt-7**

**finally:**

**stmt-9 except Y:**

**stmt-8**

**finally:**

**stmt-10**

**stmt-11**

**stmt-12**



**case-1: If there is no exception 1,2,3,4,5,6,8,9,11,12 Normal Termination**

**case-2: If an exception raised at stmt-2 and the corresponding except block matched 1,10,11,12 Normal Termination**

**case-3: If an exceptiion raised at stmt-2 and the corresponding except block not matched 1,11,Abnormal Termination**

**case-4: If an exception raised at stmt-5 and inner except block matched 1,2,3,4,7,8,9,11,12 Normal Termination**

**case-5: If an exception raised at stmt-5 and inner except block not matched but outer except block matched**

**1,2,3,4,8,10,11,12,Normal Termination**

**case-6:If an exception raised at stmt-5 and both inner and outer except blocks are not matched**

**1,2,3,4,8,11,Abnormal Termination**

**case-7: If an exception raised at stmt-7 and corresponding except block matched 1,2,3,.,.,.,8,10,11,12,Normal Termination**

**case-8: If an exception raised at stmt-7 and corresponding except block not matched 1,2,3,.,.,.,8,11,Abnormal Termination**

**case-9: If an exception raised at stmt-8 and corresponding except block matched 1,2,3,.,.,.,.,10,11,12 Normal Termination**

**case-10: If an exception raised at stmt-8 and corresponding except block not matched 1,2,3,.,.,.,.,11,Abnormal Termination**

**case-11: If an exception raised at stmt-9 and corresponding except block matched 1,2,3,.,.,.,.,8,10,11,12,Normal Termination**

**case-12: If an exception raised at stmt-9 and corresponding except block not matched 1,2,3,.,.,.,.,8,11,Abnormal Termination**

**case-13: If an exception raised at stmt-10 then it is always abonormal termination but before abnormal termination finally block(stmt-11) will be executed.**



**case-14: If an exception raised at stmt-11 or stmt-12 then it is always abnormal termination.**

**Note: If the control entered into try block then compulsary finally block will be executed. If the control not entered into try block then finally block won't be executed.**

**else block with try-except-finally:**

**We can use else block with try-except-finally blocks.**

**else block will be executed if and only if there are no exceptions inside try block.**

**try:**

**Risky Code**

**except:**

**will be executed if exception inside try**

**else: finally:**

**will be executed if there is no exception inside try**

**will be executed whether exception raised or not raised and handled or not**

**handled Eg:**

**try:**

**print("try")**

**print(10/0)--->1**

**except:**

**print("except")**

**else: finally:**

**print("else") print("finally")**

**If we comment line-1 then else block will be executed b'z there is no exception inside try. In this case the output is:**

**try else finally**

**If we are not commenting line-1 then else block won't be executed b'z there is exception inside try block. In this case output is:**



**try except finally**

**Various possible combinations of try-except-else-finally:**

1. **Whenever we are writing try block, compulsory we should write except or finally block.i.e without except or finally block we cannot write try block.**
2. **Wheneever we are writing except block, compulsory we should write try block. i.e except without try is always invalid.**
3. **Whenever we are writing finally block, compulsory we should write try block. i.e finally without try is always invalid.**
4. **We can write multiple except blocks for the same try,but we cannot write multiple finally blocks for the same try**
5. **Whenever we are writing else block compulsory except block should be there. i.e without except we cannot write else block.**
6. **In try-except-else-finally order is important.**
7. **We can define try-except-else-finally inside try,except,else and finally blocks. i.e nesting of try-except-else-finally is always possible.**

|  |  |  |
| --- | --- | --- |
| **1** | **try:**  **print("try")** |  |
| **2** | **except:**  **print("Hello")** |  |
| **3** | **else:**  **print("Hello")** |  |
| **4** | **finally:**  **print("Hello")** |  |
| **5** | **try:**  **print("try") except:**  **print("except")** | **√** |
| **6** | **try:**  **print("try")**  **finally:**  **print("finally")** | **√** |
| **7** | **try:**  **print("try")** | **√** |



|  |  |  |
| --- | --- | --- |
|  | **except:**  **print("except")**  **else:**  **print("else")** |  |
| **8** | **try:**  **print("try")**  **else:**  **print("else")** |  |
| **9** | **try:**  **print("try")**  **else:**  **print("else")**  **finally:**  **print("finally")** |  |
| **10** | **try:**  **print("try") except XXX:**  **print("except-1") except YYY:**  **print("except-2")** | **√** |
| **11** | **try:**  **print("try") except :**  **print("except-1") else:**  **print("else")**  **else:**  **print("else")** |  |
| **12** | **try:**  **print("try") except :**  **print("except-1")**  **finally:**  **print("finally")**  **finally:**  **print("finally")** |  |
| **13** | **try:**  **print("try") print("Hello")**  **except:**  **print("except")** |  |
| **14** | **try:**  **print("try") except:** |  |



|  |  |  |
| --- | --- | --- |
|  | **print("except") print("Hello")**  **except:**  **print("except")** |  |
| **15** | **try:**  **print("try") except:**  **print("except") print("Hello")**  **finally:**  **print("finally")** |  |
| **16** | **try:**  **print("try") except:**  **print("except") print("Hello")**  **else:**  **print("else")** |  |
| **17** | **try:**  **print("try") except:**  **print("except")**  **try:**  **print("try") except:**  **print("except")** | **√** |
| **18** | **try:**  **print("try") except:**  **print("except")**  **try:**  **print("try")**  **finally:**  **print("finally")** | **√** |
| **19** | **try:**  **print("try") except:**  **print("except") if 10>20:**  **print("if")**  **else:**  **print("else")** |  |
| **20** | **try:**  **print("try")** | **√** |



|  |  |  |
| --- | --- | --- |
|  | **try:**  **print("inner try") except:**  **print("inner except block") finally:**  **print("inner finally block") except:**  **print("except")** |  |
| **21** | **try:**  **print("try")**  **except:**  **print("except") try:**  **print("inner try") except:**  **print("inner except block") finally:**  **print("inner finally block")** | **√** |
| **22** | **try:**  **print("try") except:**  **print("except")**  **finally:**  **try:**  **print("inner try") except:**  **print("inner except block") finally:**  **print("inner finally block")** | **√** |
| **23** | **try:**  **print("try") except:**  **print("except")**  **try:**  **print("try")**  **else:**  **print("else")** |  |
| **24** | **try:**  **print("try") try:**  **print("inner try")**  **except:**  **print("except")** |  |



|  |  |  |
| --- | --- | --- |
| **25** | **try:**  **print("try")**  **else:**  **print("else") except:**  **print("except")**  **finally:**  **print("finally")** |  |

**Types of Exceptions:**

**In Python there are 2 types of exceptions are possible.**

1. **Predefined Exceptions**
2. **User Definded Exceptions**
3. **Predefined Exceptions:**

**Also known as in-built exceptions**

**The exceptions which are raised automatically by Python virtual machine whenver a particular event occurs, are called pre defined exceptions.**

**Eg 1: Whenever we are trying to perform Division by zero, automatically Python will raise ZeroDivisionError.**

**print(10/0)**

**Eg 2: Whenever we are trying to convert input value to int type and if input value is not int value then Python will raise ValueError automatically**

**x=int("ten")===>ValueError**

1. **User Defined Exceptions:**

**Also known as Customized Exceptions or Programatic Exceptions**

**Some time we have to define and raise exceptions explicitly to indicate that something goes wrong ,such type of exceptions are called User Defined Exceptions or Customized Exceptions**

**Programmer is responsible to define these exceptions and Python not having any idea about these. Hence we have to raise explicitly based on our requirement by using "raise" keyword.**



**Eg: InSufficientFundsException InvalidInputException TooYoungException TooOldException**

**How to Define and Raise Customized Exceptions:**

**Every exception in Python is a class that extends Exception class either directly or indirectly.**

**Syntax:**

**class classname(predefined exception class name): def init (self,arg):**

**self.msg=arg**

**Eg:**

* 1. **class TooYoungException(Exception):**

2) def init (self,arg):

3) self.msg=arg

**TooYoungException is our class name which is the child class of Exception**

**We can raise exception by using raise keyword as follows raise TooYoungException("message")**

**Eg:**

**1) class TooYoungException(Exception):**

**3)**

**self.msg=arg**

**5) class TooOldException(Exception):**

**7)**

**self.msg=arg**

**9) age=int(input("Enter Age:"))**

**11) raise TooYoungException("Plz wait some more time you will get best match soon!!!")**

**13) raise TooOldException("Your age already crossed marriage age...no chance of getting ma rriage")**

**15) print("You will get match details soon by email!!!")**

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

**16)**

**14) else:**

**12) elif age<18:**

**10) if age>60:**

**8)**

**6) def init (self,arg):**

**4)**

**2) def init (self,arg):**



|  |
| --- |
| **18) Enter Age:90** |
| **19) main .TooYoungException: Plz wait some more time you will get best match soon!!!** |
| **20)** |
| **21) D:\Python\_classes>py test.py** |
| **22) Enter Age:12** |
| **23) main .TooOldException: Your age already crossed marriage age...no chance of g** |
| **24) etting marriage** |
| **25)** |
| **26) D:\Python\_classes>py test.py** |
| **27) Enter Age:27** |
| **28) You will get match details soon by email!!!** |

**Note:**

**raise keyword is best suitable for customized exceptions but not for pre defined exceptions**



#### PYTHON LOGGING

**Logging the Exceptions:**

**It is highly recommended to store complete application flow and exceptions information to a file. This process is called logging.**

**The main advantages of logging are:**

1. **We can use log files while performing debugging**
2. **We can provide statistics like number of requests per day etc**

**To implement logging, Python provides one inbuilt module logging.**

**logging levels:**

**Depending on type of information, logging data is divided according to the following 6 levels in Python.**

**table**

1. **CRITICAL==>50==>Represents a very serious problem that needs high attention**
2. **ERROR===>40===>Represents a serious error**
3. **WARNING==>30==>Represents a warning message, some caution needed. It is alert to the programmer**
4. **INFO===>20===>Represents a message with some important information**
5. **DEBUG===>10==>Represents a message with debugging information**
6. **NOTSET==>0==>Rrepresents that the level is not set.**

**By default while executing Python program only WARNING and higher level messages will be displayed.**

**How to implement logging:**

**To perform logging, first we required to create a file to store messages and we have to specify which level messages we have to store.**

**We can do this by using basicConfig() function of logging module. logging.basicConfig(filename='log.txt',level=logging.WARNING)**



**The above line will create a file log.txt and we can store either WARNING level or higher level messages to that file.**

**After creating log file, we can write messages to that file by using the following methods. logging.debug(message)**

**logging.info(message) logging.warning(message) logging.error(message) logging.critical(message)**

**Q.Write a Python program to create a log file and write WARNING and higher level messages?**

|  |
| --- |
| **1) import logging** |
| **2) logging.basicConfig(filename='log.txt',level=logging.WARNING)** |
| **3) print("Logging Module Demo")** |
| **4) logging.debug("This is debug message")** |
| **5) logging.info("This is info message")** |
| **6) logging.warning("This is warning message")** |
| **7) logging.error("This is error message")** |
| **8) logging.critical("This is critical message")** |

**log.txt:**

1) WARNING:root:This is warning message

2) ERROR:root:This is error message

3) CRITICAL:root:This is critical message

**Note:**

**In the above program only WARNING and higher level messages will be written to log file. If we set level as DEBUG then all messages will be written to log file.**

|  |
| --- |
| **1) import logging** |
| **2) logging.basicConfig(filename='log.txt',level=logging.DEBUG)** |
| **3) print("Logging Module Demo")** |
| **4) logging.debug("This is debug message")** |
| **5) logging.info("This is info message")** |
| **6) logging.warning("This is warning message")** |
| **7) logging.error("This is error message")** |
| **8) logging.critical("This is critical message")** |

**log.txt:**

|  |
| --- |
| **1) DEBUG:root:This is debug message** |
| **2) INFO:root:This is info message** |



3) WARNING:root:This is warning message

4) ERROR:root:This is error message

5) CRITICAL:root:This is critical message

**Note: We can format log messages to include date and time, ip address of the client etc at advanced level.**

**How to write Python program exceptions to the log file:**

**By using the following function we can write exceptions information to the log file. logging.exception(msg)**

**Q. Python Program to write exception information to the log file**

|  |
| --- |
| **1) import logging** |
| **2) logging.basicConfig(filename='mylog.txt',level=logging.INFO)** |
| **3) logging.info("A New request Came:")** |
| **4) try:** |
| **5) x=int(input("Enter First Number: "))** |
| **6) y=int(input("Enter Second Number: "))** |
| **7) print(x/y)** |
| **8) except ZeroDivisionError as msg:** |
| **9) print("cannot divide with zero")** |
| **10) logging.exception(msg)** |
| **11) except ValueError as msg:** |
| **12) print("Enter only int values")** |
| **13) logging.exception(msg)** |
| **14) logging.info("Request Processing Completed")** |
| **15)** |
| **16)** |
| **17) D:\Python\_classes>py test.py** |
| **18) Enter First Number: 10** |
| **19) Enter Second Number: 2** |
| **20) 5.0** |
| **21)** |
| **22) D:\Python\_classes>py test.py** |
| **23) Enter First Number: 10** |
| **24) Enter Second Number: 0** |
| **25) cannot divide with zero** |
| **26)** |
| **27) D:\Python\_classes>py test.py** |
| **28) Enter First Number: 10** |
| **29) Enter Second Number: ten** |
| **30) Enter only int values** |



**mylog.txt:**

|  |
| --- |
| **1) INFO:root:A New request Came:** |
| **2) INFO:root:Request Processing Completed** |
| **3) INFO:root:A New request Came:** |
| **4) ERROR:root:division by zero** |
| **5) Traceback (most recent call last):** |
| **6) File "test.py", line 7, in <module>** |
| **7) print(x/y)** |
| **8) ZeroDivisionError: division by zero** |
| **9) INFO:root:Request Processing Completed** |
| **10) INFO:root:A New request Came:** |
| **11) ERROR:root:invalid literal for int() with base 10: 'ten'** |
| **12) Traceback (most recent call last):** |
| **13) File "test.py", line 6, in <module>** |
| **14) y=int(input("Enter Second Number: "))** |
| **15) ValueError: invalid literal for int() with base 10: 'ten'** |
| **16) INFO:root:Request Processing Completed** |



**PYTHON DEBUGGING BY USING ASSERTIONS**

**Debugging Python Program by using assert keyword:**

**The process of identifying and fixing the bug is called debugging.**

**Very common way of debugging is to use print() statement. But the problem with the print() statement is after fixing the bug,compulsory we have to delete the extra added print() statments,otherwise these will be executed at runtime which creates performance problems and disturbs console output.**

**To overcome this problem we should go for assert statement. The main advantage of assert statement over print() statement is after fixing bug we are not required to delete assert statements. Based on our requirement we can enable or disable assert statements.**

**Hence the main purpose of assertions is to perform debugging. Usully we can perform debugging either in development or in test environments but not in production environment. Hence assertions concept is applicable only for dev and test environments but not for production environment.**

**Types of assert statements:**

**There are 2 types of assert statements**

1. **Simple Version**
2. **Augmented Version**
3. **Simple Version:**

**assert conditional\_expression**

1. **Augmented Version:**

**assert conditional\_expression,message**

**conditional\_expression will be evaluated and if it is true then the program will be continued.**

**If it is false then the program will be terminated by raising AssertionError.**

**By seeing AssertionError, programmer can analyze the code and can fix the problem. Eg:**

|  |
| --- |
| **1) def squareIt(x):** |
| **2) return x\*\*x** |



|  |
| --- |
| **3) assert squareIt(2)==4,"The square of 2 should be 4"** |
| **4) assert squareIt(3)==9,"The square of 3 should be 9"** |
| **5) assert squareIt(4)==16,"The square of 4 should be 16"** |
| **6) print(squareIt(2))** |
| **7) print(squareIt(3))** |
| **8) print(squareIt(4))** |
| **9)** |
| **10) D:\Python\_classes>py test.py** |
| **11) Traceback (most recent call last):** |
| **12) File "test.py", line 4, in <module>** |
| **13) assert squareIt(3)==9,"The square of 3 should be 9"** |
| **14) AssertionError: The square of 3 should be 9** |
| **15)** |
| **16) def squareIt(x):** |
| **17) return x\*x** |
| **18) assert squareIt(2)==4,"The square of 2 should be 4"** |
| **19) assert squareIt(3)==9,"The square of 3 should be 9"** |
| **20) assert squareIt(4)==16,"The square of 4 should be 16"** |
| **21) print(squareIt(2))** |
| **22) print(squareIt(3))** |
| **23) print(squareIt(4))** |
| **24)** |
| **25) Output** |
| **26) 4** |
| **27) 9** |
| **28) 16** |

**Exception Handling vs Assertions:**

**Assertions concept can be used to alert programmer to resolve development time errors. Exception Handling can be used to handle runtime errors.**



**Python's Object Oriented Programming (OOPs)**

**What is Class:**

⚽ In Python every thing is an object. To create objects we required some Model or Plan or Blue print, which is nothing but class.

⚽ We can write a class to represent properties (attributes) and actions (behaviour) of object.

⚽ Properties can be represented by variables

⚽ Actions can be represented by Methods.

⚽ Hence class contains both variables and methods.

**How to Define a class?**

We can define a class by using class keyword.

**Syntax:**

class className:

''' documenttation string '''

variables:instance variables,static and local variables methods: instance methods,static methods,class methods

Documentation string represents description of the class. Within the class doc string is always optional. We can get doc string by using the following 2 ways.

1. **print(classname. doc )**
2. **help(classname)**

**Example:**

|  |
| --- |
| **1) class Student:** |
| **2) ''''' This is student class with required data'''** |
| **3) print(Student. doc )** |
| **4) help(Student)** |

Within the Python class we can represent data by using variables. There are 3 types of variables are allowed.

1. **Instance Variables (Object Level Variables)**
2. **Static Variables (Class Level Variables)**
3. **Local variables (Method Level Variables)**

Within the Python class, we can represent operations by using methods. The following are various types of allowed methods



1. **Instance Methods**
2. **Class Methods**
3. **Static Methods**

**Example for class:**

**1) class Student:**

**3) def init (self):**

**5)**

**self.age=40**

**7)**

**9)**

**print("Hello I am :",self.name)**

**11)**

**print("My Marks are:",self.marks)**

**print("My Age is:",self.age)**

**10)**

**8) def talk(self):**

**self.marks=80**

**6)**

**self.name='durga'**

**4)**

**2) '''''Developed by durga for python demo'''**

**What is Object:**

Pysical existence of a class is nothing but object. We can create any number of objects for a class.

**Syntax to create object: referencevariable = classname()**

**Example: s = Student()**

**What is Reference Variable:**

The variable which can be used to refer object is called reference variable. By using reference variable, we can access properties and methods of object.

Program: Write a Python program to create a Student class and Creates an object to it. Call the method talk() to display student details

|  |
| --- |
| **1) class Student:** |
| **2)** |
| **3) def init (self,name,rollno,marks):** |
| **4) self.name=name** |
| **5) self.rollno=rollno** |
| **6) self.marks=marks** |
| **7)** |
| **8) def talk(self):** |
| **9) print("Hello My Name is:",self.name)** |
| **10) print("My Rollno is:",self.rollno)** |
| **11) print("My Marks are:",self.marks)** |
| **12)** |



|  |
| --- |
| **13) s1=Student("Durga",101,80)** |
| **14) s1.talk()** |

Output: D:\durgaclasses>py test.py Hello My Name is: Durga My Rollno is: 101

My Marks are: 80

**Self variable:**

self is the default variable which is always pointing to current object (like this keyword in Java) By using self we can access instance variables and instance methods of object.

**Note:**

1. **self should be first parameter inside constructor def init (self):**
2. **self should be first parameter inside instance methods def talk(self):**

**Constructor Concept:**

☕ Constructor is a special method in python.

☕ The name of the constructor should be init (self)

☕ Constructor will be executed automatically at the time of object creation.

☕ The main purpose of constructor is to declare and initialize instance variables.

☕ Per object constructor will be exeucted only once.

☕ Constructor can take atleast one argument(atleast self)

☕ Constructor is optional and if we are not providing any constructor then python will provide default constructor.

**Example:**

**1) def init (self,name,rollno,marks):**

**3)**

**self.rollno=rollno**

**self.marks=marks**

**4)**

**self.name=name**

**2)**

**Program to demonistrate constructor will execute only once per object:**

**1) class Test:**

**3) def init (self): 5)**

**print("Constructor exeuction...")**

**4)**

**2)**



|  |
| --- |
| **6) def m1(self):** |
| **7) print("Method execution...")** |
| **8)** |
| **9) t1=Test()** |
| **10) t2=Test()** |
| **11) t3=Test()** |
| **12) t1.m1()** |

**Output**

Constructor exeuction... Constructor exeuction... Constructor exeuction... Method execution...

**Program:**

**1) class Student:**

**15) s2.display()**

**14) s2=Student("Sunny",102,100)**

**13) s1.display()**

**12) s1=Student("Durga",101,80)**

11)

**10) print("Student Name:{}\nRollno:{} \nMarks:{}".format(self.name,self.rollno,self.marks)**

**)**

**9) def display(self):**

**8)**

**self.marks=z**

**7)**

**self.rollno=y**

**6)**

**self.name=x**

**5)**

**4) def init (self,x,y,z):**

**3) ''''' This is student class with required data'''**

**2)**

**Output**

Student Name:Durga Rollno:101

Marks:80

Student Name:Sunny Rollno:102 Marks:100



**Differences between Methods and Constructors:**

|  |  |
| --- | --- |
| **Method** | **Constructor** |
| **1. Name of method can be any name** | **1. Constructor name should be always init** |
| **2. Method will be executed if we call that**  **method** | **2. Constructor will be executed automatically at**  **the time of object creation.** |
| **3. Per object, method can be called any number**  **of times.** | **3. Per object, Constructor will be executed only**  **once** |
| **4. Inside method we can write business logic** | **4. Inside Constructor we have to declare and**  **initialize instance variables** |

**Types of Variables:**

Inside Python class 3 types of variables are allowed.

1. **Instance Variables (Object Level Variables)**
2. **Static Variables (Class Level Variables)**
3. **Local variables (Method Level Variables)**

**1. Instance Variables:**

If the value of a variable is varied from object to object, then such type of variables are called instance variables.

For every object a separate copy of instance variables will be created.

**Where we can declare Instance variables:**

1. **Inside Constructor by using self variable**
2. **Inside Instance Method by using self variable**
3. **Outside of the class by using object reference variable**
4. **Inside Constructor by using self variable:**

We can declare instance variables inside a constructor by using self keyword. Once we creates object, automatically these variables will be added to the object.

**Example:**

|  |
| --- |
| **1) class Employee:** |
| **2)** |
| **3) def init (self):** |
| **4) self.eno=100** |
| **5) self.ename='Durga'** |
| **6) self.esal=10000** |
| **7)** |
| **8) e=Employee()** |



9) print(e. dict )

Output: {'eno': 100, 'ename': 'Durga', 'esal': 10000}

1. **Inside Instance Method by using self variable:**

We can also declare instance variables inside instance method by using self variable. If any instance variable declared inside instance method, that instance variable will be added once we call taht method.

**Example:**

|  |
| --- |
| **1) class Test:** |
| **2)** |
| **3) def init (self):** |
| **4) self.a=10** |
| **5) self.b=20** |
| **6)** |
| **7) def m1(self):** |
| **8) self.c=30** |
| **9)** |
| **10) t=Test()** |
| **11) t.m1()** |
| **12) print(t. dict )** |

**Output**

{'a': 10, 'b': 20, 'c': 30}

1. **Outside of the class by using object reference variable:**

We can also add instance variables outside of a class to a particular object.

**1) class Test:**

**3) def init (self):**

**5)**

**self.b=20**

**7) def m1(self):**

**9)**

**11) t.m1()**

**13) print(t. dict )**

**12) t.d=40**

**10) t=Test()**

**self.c=30**

**8)**

**6)**

**self.a=10**

**4)**

**2)**

Output {'a': 10, 'b': 20, 'c': 30, 'd': 40}



**How to access Instance variables:**

We can access instance variables with in the class by using self variable and outside of the class by using object reference.

**1) class Test:**

**3) def init (self):**

**5)**

**self.b=20**

**7) def display(self):**

**9)**

**print(self.b)**

**11) t=Test()**

**13) print(t.a,t.b)**

**12) t.display()**

**10)**

**print(self.a)**

**8)**

**6)**

**self.a=10**

**4)**

**2)**

**Output**

10

20

10 20

**How to delete instance variable from the object:**

1. **Within a class we can delete instance variable as follows**

del self.variableName

1. **From outside of class we can delete instance variables as follows**

del objectreference.variableName

**Example:**

* 1. **class Test:**
  2. **def init (self):**
  3. **self.a=10**
  4. **self.b=20**
  5. **self.c=30**
  6. **self.d=40**
  7. **def m1(self):**
  8. **del self.d 9)**

1. **t=Test()**
2. **print(t. dict )**



|  |
| --- |
| **12) t.m1()** |
| **13) print(t. dict )** |
| **14) del t.c** |
| **15) print(t. dict )** |

**Output**

{'a': 10, 'b': 20, 'c': 30, 'd': 40}

{'a': 10, 'b': 20, 'c': 30}

{'a': 10, 'b': 20}

Note: The instance variables which are deleted from one object,will not be deleted from other objects.

**Example:**

**1) class Test:**

**3)**

**self.a=10**

**5)**

**self.c=30**

**7)**

**9) t1=Test()**

**11) del t1.a**

**13) print(t2. dict )**

**12) print(t1. dict )**

**10) t2=Test()**

**8)**

**self.d=40**

**6)**

**self.b=20**

**4)**

**2) def init (self):**

**Output**

{'b': 20, 'c': 30, 'd': 40}

{'a': 10, 'b': 20, 'c': 30, 'd': 40}

If we change the values of instance variables of one object then those changes won't be reflected to the remaining objects, because for every object we are separate copy of instance variables are available.

**Example:**

|  |
| --- |
| **1) class Test:** |
| **2) def init (self):** |
| **3) self.a=10** |
| **4) self.b=20** |
| **5)** |
| **6) t1=Test()** |
| **7) t1.a=888** |
| **8) t1.b=999** |
| **9) t2=Test()** |
| **10) print('t1:',t1.a,t1.b)** |



11) print('t2:',t2.a,t2.b)

**Output**

t1: 888 999

t2: 10 20

**1. Static variables:**

If the value of a variable is not varied from object to object, such type of variables we have to declare with in the class directly but outside of methods. Such type of variables are called Static variables.

For total class only one copy of static variable will be created and shared by all objects of that class.

We can access static variables either by class name or by object reference. But recommended to use class name.

**Instance Variable vs Static Variable:**

Note: In the case of instance variables for every object a seperate copy will be created,but in the case of static variables for total class only one copy will be created and shared by every object of that class.

**1) class Test:**

**3) def init (self): 5)**

**7) t2=Test()**

**9) print('t2:',t2.x,t2.y)**

**11) t1.y=999**

**13) print('t2:',t2.x,t2.y)**

**12) print('t1:',t1.x,t1.y)**

**10) Test.x=888**

**8) print('t1:',t1.x,t1.y)**

**6) t1=Test()**

**self.y=20**

**4)**

**2) x=10**

**Output**

t1: 10 20

t2: 10 20

t1: 888 999

t2: 888 20



**Various places to declare static variables:**

1. **In general we can declare within the class directly but from out side of any method**
2. **Inside constructor by using class name**
3. **Inside instance method by using class name**
4. **Inside classmethod by using either class name or cls variable**
5. **Inside static method by using class name**

|  |
| --- |
| **1) class Test:** |
| **2) a=10** |
| **3) def init (self):** |
| **4) Test.b=20** |
| **5) def m1(self):** |
| **6) Test.c=30** |
| **7) @classmethod** |
| **8) def m2(cls):** |
| **9) cls.d1=40** |
| **10) Test.d2=400** |
| **11) @staticmethod** |
| **12) def m3():** |
| **13) Test.e=50** |
| **14) print(Test. dict )** |
| **15) t=Test()** |
| **16) print(Test. dict )** |
| **17) t.m1()** |
| **18) print(Test. dict )** |
| **19) Test.m2()** |
| **20) print(Test. dict )** |
| **21) Test.m3()** |
| **22) print(Test. dict )** |
| **23) Test.f=60** |
| **24) print(Test. dict )** |

**How to access static variables:**

1. **inside constructor: by using either self or classname**
2. **inside instance method: by using either self or classname**
3. **inside class method: by using either cls variable or classname**
4. **inside static method: by using classname**
5. **From outside of class: by using either object reference or classnmae**

**1) class Test:**

**3) def init (self):**

**5)**

**print(Test.a)**

**7)**

**print(self.a)**

**6) def m1(self):**

**print(self.a)**

**4)**

**2) a=10**



|  |
| --- |
| **8) print(Test.a)** |
| **9) @classmethod** |
| **10) def m2(cls):** |
| **11) print(cls.a)** |
| **12) print(Test.a)** |
| **13) @staticmethod** |
| **14) def m3():** |
| **15) print(Test.a)** |
| **16) t=Test()** |
| **17) print(Test.a)** |
| **18) print(t.a)** |
| **19) t.m1()** |
| **20) t.m2()** |
| **21) t.m3()** |

**Where we can modify the value of static variable:**

Anywhere either with in the class or outside of class we can modify by using classname. But inside class method, by using cls variable.

**Example:**

**1) class Test:**

**3) @classmethod**

**5)**

**cls.a=888**

**7) def m2():**

**9) print(Test.a)**

**11) print(Test.a)**

**13) print(Test.a)**

**12) Test.m2()**

**10) Test.m1()**

**Test.a=999**

**8)**

**6) @staticmethod**

**4) def m1(cls):**

**2) a=777**

**Output**

777

888

999



\*\*\*\*\*

**If we change the value of static variable by using either self or object reference variable:**

If we change the value of static variable by using either self or object reference variable, then the value of static variable won't be changed,just a new instance variable with that name will be added to that particular object.

**Example 1:**

|  |
| --- |
| **1) class Test:** |
| **2) a=10** |
| **3) def m1(self):** |
| **4) self.a=888** |
| **5) t1=Test()** |
| **6) t1.m1()** |
| **7) print(Test.a)** |
| **8) print(t1.a)** |

**Output**

10

888

**Example:**

**1) class Test:**

**3) def init (self): 5)**

**7) t2=Test()**

**9) print('t2:',t2.x,t2.y)**

**11) t1.y=999**

**13) print('t2:',t2.x,t2.y)**

**12) print('t1:',t1.x,t1.y)**

**10) t1.x=888**

**8) print('t1:',t1.x,t1.y)**

**6) t1=Test()**

**self.y=20**

**4)**

**2) x=10**

**Output**

t1: 10 20

t2: 10 20

t1: 888 999

t2: 10 20



**Example:**

|  |
| --- |
| **1) class Test:** |
| **2) a=10** |
| **3) def init (self):** |
| **4) self.b=20** |
| **5) t1=Test()** |
| **6) t2=Test()** |
| **7) Test.a=888** |
| **8) t1.b=999** |
| **9) print(t1.a,t1.b)** |
| **10) print(t2.a,t2.b)** |

**Output**

888 999

888 20

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

**1) class Test:**

**3) def init (self):**

**5) def m1(self):**

**7)**

**self.b=999**

**9) t1=Test()**

**11) t1.m1()**

13) **print(t2.a,t2.b)**

**12) print(t1.a,t1.b)**

**10) t2=Test()**

**8)**

**self.a=888**

**6)**

**self.b=20**

**4)**

**2) a=10**

**Output**

888 999

10 20

**Example:**

**1) class Test:**

**3) def init (self):**

**5) @classmethod**

**7)**

**cls.a=888**

**9)**

**11) t2=Test()**

**10) t1=Test()**

**cls.b=999**

**8)**

**6) def m1(cls):**

**self.b=20**

**4)**

**2) a=10**



|  |
| --- |
| **12) t1.m1()** |
| **13) print(t1.a,t1.b)** |
| **14) print(t2.a,t2.b)** |
| **15) print(Test.a,Test.b)** |

**Output**

888 20

888 20

888 999

**How to delete static variables of a class:**

We can delete static variables from anywhere by using the following syntax

del classname.variablename

But inside classmethod we can also use cls variable

del cls.variablename

* 1. **class Test:**

2) a=10

1. **@classmethod**
2. **def m1(cls):**
3. **del cls.a**
4. **Test.m1()**
5. **print(Test. dict )**

**Example:**

1) class Test:

2) a=10

1. **def init (self):**
2. **Test.b=20**
3. **del Test.a**
4. **def m1(self):**
5. **Test.c=30**
6. **del Test.b**
7. **@classmethod**
8. **def m2(cls):**
9. **cls.d=40**
10. **del Test.c**
11. **@staticmethod**
12. **def m3():**
13. **Test.e=50**
14. **del Test.d**
15. **print(Test. dict )**
16. **t=Test()**



1. **print(Test. dict )**

20) t.m1()

21) print(Test. dict )

22) Test.m2()

23) print(Test. dict )

24) Test.m3()

25) print(Test. dict )

26) Test.f=60

27) print(Test. dict )

28) del Test.e

29) print(Test. dict )

\*\*\*\*

Note: By using object reference variable/self we can read static variables, but we cannot modify or delete.

If we are trying to modify, then a new instance variable will be added to that particular object. t1.a = 70

If we are trying to delete then we will get error.

**Example:**

1) class Test:

2) a=10

3)

4) t1=Test()

5) del t1.a ===>AttributeError: a

We can modify or delete static variables only by using classname or cls variable.

|  |
| --- |
| **1) import sys** |
| **2) class Customer:** |
| **3) ''''' Customer class with bank operations.. '''** |
| **4) bankname='DURGABANK'** |
| **5) def init (self,name,balance=0.0):** |
| **6) self.name=name** |
| **7) self.balance=balance** |
| **8) def deposit(self,amt):** |
| **9) self.balance=self.balance+amt** |
| **10) print('Balance after deposit:',self.balance)** |
| **11) def withdraw(self,amt):** |
| **12) if amt>self.balance:** |
| **13) print('Insufficient Funds..cannot perform this operation')** |
| **14) sys.exit()** |
| **15) self.balance=self.balance-amt** |
| **16) print('Balance after withdraw:',self.balance)** |
| **17)** |
| **18) print('Welcome to',Customer.bankname)** |



|  |
| --- |
| **19) name=input('Enter Your Name:')** |
| **20) c=Customer(name)** |
| **21) while True:** |
| **22) print('d-Deposit \nw-Withdraw \ne-exit')** |
| **23) option=input('Choose your option:')** |
| **24) if option=='d' or option=='D':** |
| **25) amt=float(input('Enter amount:'))** |
| **26) c.deposit(amt)** |
| **27) elif option=='w' or option=='W':** |
| **28) amt=float(input('Enter amount:'))** |
| **29) c.withdraw(amt)** |
| **30) elif option=='e' or option=='E':** |
| **31) print('Thanks for Banking')** |
| **32) sys.exit()** |
| **33) else:** |
| **34) print('Invalid option..Plz choose valid option')** |

output: D:\durga\_classes>py test.py Welcome to DURGABANK Enter Your Name:Durga

d-Deposit

w-Withdraw e-exit

Choose your option:d Enter amount:10000

Balance after deposit: 10000.0 d-Deposit

w-Withdraw e-exit

Choose your option:d Enter amount:20000

Balance after deposit: 30000.0 d-Deposit

w-Withdraw e-exit

Choose your option:w Enter amount:2000

Balance after withdraw: 28000.0 d-Deposit

w-Withdraw e-exit

Choose your option:r

Invalid option..Plz choose valid option d-Deposit

w-Withdraw e-exit

Choose your option:e Thanks for Banking



**17**

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

**nd**

**Local variables:**

Sometimes to meet temporary requirements of programmer,we can declare variables inside a method directly,such type of variables are called local variable or temporary variables.

Local variables will be created at the time of method execution and destroyed once method completes.

Local variables of a method cannot be accessed from outside of method.

**Example:**

|  |
| --- |
| **1) class Test:** |
| **2) def m1(self):** |
| **3) a=1000** |
| **4) print(a)** |
| **5) def m2(self):** |
| **6) b=2000** |
| **7) print(b)** |
| **8) t=Test()** |
| **9) t.m1()** |
| **10) t.m2()** |

**Output**

1000

2000

**Example 2:**

**1) class Test:**

**3)**

**a=1000**

**5) def m2(self):**

**7)**

**print(a) #NameError: name 'a' is not defined**

**9) t=Test()**

**11) t.m2()**

**10) t.m1()**

**print(b)**

**8)**

**b=2000**

**6)**

**print(a)**

**4)**

**2) def m1(self):**



**Types of Methods:**

Inside Python class 3 types of methods are allowed

1. **Instance Methods**
2. **Class Methods**
3. **Static Methods**
4. **Instance Methods:**

Inside method implementation if we are using instance variables then such type of methods are called instance methods.

Inside instance method declaration,we have to pass self variable.

def m1(self):

By using self variable inside method we can able to access instance variables.

Within the class we can call instance method by using self variable and from outside of the class we can call by using object reference.

* 1. **class Student:**
  2. **def init (self,name,marks):**
  3. **self.name=name**
  4. **self.marks=marks**
  5. **def display(self):**
  6. **print('Hi',self.name)**
  7. **print('Your Marks are:',self.marks)**
  8. **def grade(self):**
  9. **if self.marks>=60:**
  10. **print('You got First Grade')**
  11. **elif self.marks>=50:**
  12. **print('Yout got Second Grade')**
  13. **elif self.marks>=35:**
  14. **print('You got Third Grade')**
  15. **else:**
  16. **print('You are Failed')**
  17. **n=int(input('Enter number of students:'))**
  18. **for i in range(n):**
  19. **name=input('Enter Name:')**
  20. **marks=int(input('Enter Marks:'))**
  21. **s= Student(name,marks)**
  22. **s.display()**
  23. **s.grade()**
  24. **print()**



**ouput:**

D:\durga\_classes>py test.py Enter number of students:2 Enter Name:Durga

Enter Marks:90 Hi Durga

Your Marks are: 90 You got First Grade

Enter Name:Ravi Enter Marks:12 Hi Ravi

Your Marks are: 12 You are Failed

**Setter and Getter Methods:**

We can set and get the values of instance variables by using getter and setter methods.

**Setter Method:**

setter methods can be used to set values to the instance variables. setter methods also known as mutator methods.

**syntax:**

def setVariable(self,variable): self.variable=variable

**Example:**

def setName(self,name): self.name=name

**Getter Method:**

Getter methods can be used to get values of the instance variables. Getter methods also known as accessor methods.

**syntax:**

def getVariable(self): return self.variable

**Example:**

def getName(self): return self.name



**Demo Program:**

|  |
| --- |
| **1) class Student:** |
| **2) def setName(self,name):** |
| **3) self.name=name** |
| **4)** |
| **5) def getName(self):** |
| **6) return self.name** |
| **7)** |
| **8) def setMarks(self,marks):** |
| **9) self.marks=marks** |
| **10)** |
| **11) def getMarks(self):** |
| **12) return self.marks** |
| **13)** |
| **14) n=int(input('Enter number of students:'))** |
| **15) for i in range(n):** |
| **16) s=Student()** |
| **17) name=input('Enter Name:')** |
| **18) s.setName(name)** |
| **19) marks=int(input('Enter Marks:'))** |
| **20) s.setMarks(marks)** |
| **21)** |
| **22) print('Hi',s.getName())** |
| **23) print('Your Marks are:',s.getMarks())** |
| **24) print()** |

output: D:\python\_classes>py test.py Enter number of students:2 Enter Name:Durga

Enter Marks:100 Hi Durga

Your Marks are: 100

Enter Name:Ravi Enter Marks:80 Hi Ravi

Your Marks are: 80

1. **Class Methods:**

Inside method implementation if we are using only class variables (static variables), then such type of methods we should declare as class method.

We can declare class method explicitly by using @classmethod decorator. For class method we should provide cls variable at the time of declaration



We can call classmethod by using classname or object reference variable.

**Demo Program:**

* 1. **class Animal:**

2) legs=4

3) @classmethod

4) def walk(cls,name):

5) print('{} walks with {} legs...'.format(name,cls.legs))

6) Animal.walk('Dog')

7) Animal.walk('Cat')

Output D:\python\_classes>py test.py Dog walks with 4 legs...

Cat walks with 4 legs...

**Program to track the number of objects created for a class:**

**1) class Test:**

**3) def init (self):**

5)

**@classmethod**

**7)**

**print('The number of objects created for test class:',cls.count)**

**9) t1=Test()**

**11) Test.noOfObjects()**

**13) t4=Test()**

**15) Test.noOfObjects()**

**14) t5=Test()**

**12) t3=Test()**

**10) t2=Test()**

**8)**

**6) def noOfObjects(cls):**

**Test.count =Test.count+1**

**4)**

**2) count=0**

1. **Static Methods:**

In general these methods are general utility methods.

Inside these methods we won't use any instance or class variables. Here we won't provide self or cls arguments at the time of declaration.

We can declare static method explicitly by using @staticmethod decorator We can access static methods by using classname or object reference

|  |
| --- |
| **1) class DurgaMath:** |
| **2)** |
| **3) @staticmethod** |
| **4) def add(x,y):** |



**5)**

**print('The Sum:',x+y)**

**7) @staticmethod**

**9)**

**print('The Product:',x\*y)**

**11) @staticmethod**

**13)**

**print('The average:',(x+y)/2)**

**15) DurgaMath.add(10,20)**

**17) DurgaMath.average(10,20)**

**16) DurgaMath.product(10,20)**

**14)**

**12) def average(x,y):**

**10)**

**8) def product(x,y):**

**6)**

**Output**

The Sum: 30

The Product: 200

The average: 15.0

Note: In general we can use only instance and static methods.Inside static method we can access class level variables by using class name.

class methods are most rarely used methods in python.

**Passing members of one class to another class:**

We can access members of one class inside another class.

**1) class Employee:**

**3)**

**self.eno=eno**

**5)**

**self.esal=esal**

**7)**

**print('Employee Number:',self.eno)**

**9)**

**print('Employee Salary:',self.esal)**

**11) def modify(emp):**

**13)**

**emp.display()**

15) **Test.modify(e)**

**14) e=Employee(100,'Durga',10000)**

**emp.esal=emp.esal+10000**

**12)**

**10) class Test:**

**print('Employee Name:',self.ename)**

**8)**

**6) def display(self):**

**self.ename=ename**

**4)**

**2) def init (self,eno,ename,esal):**

Output D:\python\_classes>py test.py Employee Number: 100 Employee Name: Durga



Employee Salary: 20000

In the above application, Employee class members are available to Test class.

**Inner classes:**

Sometimes we can declare a class inside another class,such type of classes are called inner classes.

Without existing one type of object if there is no chance of existing another type of object,then we should go for inner classes.

Example: Without existing Car object there is no chance of existing Engine object. Hence Engine class should be part of Car class.

class Car:

.....

class Engine:

......

Example: Without existing university object there is no chance of existing Department object class University:

.....

class Department:

......

**eg3:**

Without existing Human there is no chance of existin Head. Hence Head should be part of Human.

class Human:

class Head:

Note: Without existing outer class object there is no chance of existing inner class object. Hence inner class object is always associated with outer class object.

**Demo Program-1:**

**1) class Outer:**

**3)**

**print("outer class object creation")**

**5)**

**def init (self):**

**7)**

**def m1(self):**

**9) o=Outer()**

**11) i.m1()**

**10) i=o.Inner()**

**print("inner class method")**

**8)**

**print("inner class object creation")**

**6)**

**4) class Inner:**

**2) def init (self):**



**Output**

outer class object creation inner class object creation inner class method

Note: The following are various possible syntaxes for calling inner class method 1.

o=Outer() i=o.Inner() i.m1()

2.

i=Outer().Inner() i.m1()

3. Outer().Inner().m1()

**Demo Program-2:**

**1) class Person:**

**3)**

**self.name='durga'**

**5) def display(self):**

**7) class Dob:**

**9)**

**self.dd=10**

**11)**

**self.yy=1947**

**13)**

**print('Dob={}/{}/{}'.format(self.dd,self.mm,self.yy))**

**15) p.display()**

**17) x.display()**

**16) x=p.db**

**14) p=Person()**

**def display(self):**

**12)**

**self.mm=5**

**10)**

**def init (self):**

**8)**

**print('Name:',self.name)**

**6)**

**self.db=self.Dob()**

**4)**

**2) def init (self):**

**Output**

Name: durga Dob=10/5/1947

**Demo Program-3:**

Inside a class we can declare any number of inner classes.

**1) class Human:**

**3) def init (self):**

**2)**



|  |
| --- |
| **4) self.name = 'Sunny'** |
| **5) self.head = self.Head()** |
| **6) self.brain = self.Brain()** |
| **7) def display(self):** |
| **8) print("Hello..",self.name)** |
| **9)** |
| **10) class Head:** |
| **11) def talk(self):** |
| **12) print('Talking...')** |
| **13)** |
| **14) class Brain:** |
| **15) def think(self):** |
| **16) print('Thinking...')** |
| 17) |
| **18) h=Human()** |
| **19) h.display()** |
| **20) h.head.talk()** |
| **21) h.brain.think()** |

**Output Hello.. Sunny Talking...**

Thinking...

**Garbage Collection:**

In old languages like C++, programmer is responsible for both creation and destruction of objects.Usually programmer taking very much care while creating object, but neglecting destruction of useless objects. Because of his neglectance, total memory can be filled with useless objects which creates memory problems and total application will be down with Out of memory error.

But in Python, We have some assistant which is always running in the background to destroy useless objects.Because this assistant the chance of failing Python program with memory problems is very less. This assistant is nothing but Garbage Collector.

Hence the main objective of Garbage Collector is to destroy useless objects.

If an object does not have any reference variable then that object eligible for Garbage Collection.

**How to enable and disable Garbage Collector in our program:**

By default Gargbage collector is enabled, but we can disable based on our requirement. In this context we can use the following functions of gc module.

1. **gc.isenabled()**

Returns True if GC enabled



1. **gc.disable()**

To disable GC explicitly

1. **gc.enable()**

To enable GC explicitly

**Example:**

|  |
| --- |
| **1) import gc** |
| **2) print(gc.isenabled())** |
| **3) gc.disable()** |
| **4) print(gc.isenabled())** |
| **5) gc.enable()** |
| **6) print(gc.isenabled())** |

**Output True False True**

**Destructors:**

Destructor is a special method and the name should be del

Just before destroying an object Garbage Collector always calls destructor to perform clean up activities (Resource deallocation activities like close database connection etc).

Once destructor execution completed then Garbage Collector automatically destroys that object.

Note: The job of destructor is not to destroy object and it is just to perform clean up activities.

**Example:**

**1) import time**

**3) def init (self):**

**5) def del (self): 7)**

**9) t1=None**

**11) print("End of application")**

**10) time.sleep(5)**

**8) t1=Test()**

**print("Fulfilling Last Wish and performing clean up activities...")**

**6)**

**print("Object Initialization...")**

**4)**

**2) class Test:**

**Output**

Object Initialization...

Fulfilling Last Wish and performing clean up activities... End of application



**Note:**

If the object does not contain any reference variable then only it is eligible fo GC. ie if the reference count is zero then only object eligible for GC

**Example:**

|  |
| --- |
| **1) import time** |
| **2) class Test:** |
| **3) def init (self):** |
| **4) print("Constructor Execution...")** |
| **5) def del (self):** |
| **6) print("Destructor Execution...")** |
| **7)** |
| **8) t1=Test()** |
| **9) t2=t1** |
| **10) t3=t2** |
| **11) del t1** |
| **12) time.sleep(5)** |
| **13) print("object not yet destroyed after deleting t1")** |
| **14) del t2** |
| **15) time.sleep(5)** |
| **16) print("object not yet destroyed even after deleting t2")** |
| **17) print("I am trying to delete last reference variable...")** |
| 18) **del t3** |

**Example:**

**1) import time**

**3) def init (self):**

**5) def del (self): 7)**

**9) del list**

**11) print("End of application")**

**10) time.sleep(5)**

**8) list=[Test(),Test(),Test()]**

**print("Destructor Execution...")**

**6)**

**print("Constructor Execution...")**

**4)**

**2) class Test:**

**Output**

Constructor Execution... Constructor Execution... Constructor Execution... Destructor Execution... Destructor Execution... Destructor Execution... End of application



**How to find the number of references of an object:**

sys module contains getrefcount() function for this purpose.

sys.getrefcount(objectreference)

**Example:**

* 1. **import sys**
  2. **class Test:**
  3. **pass**
  4. **t1=Test() 5) t2=t1**

6) t3=t1

7) t4=t1

8) print(sys.getrefcount(t1))

**Output 5**

Note: For every object, Python internally maintains one default reference variable self.



**Polymorphism**

Poly means many. Morphs means forms. Polymorphism means 'Many Forms'.

Eg1: Yourself is best example of polymorphism.In front of Your parents You will have one type of behaviour and with friends another type of behaviour.Same person but different behaviours at different places,which is nothing but polymorphism.

Eg2: + operator acts as concatenation and arithmetic addition

Eg3: \* operator acts as multiplication and repetition operator

Eg4: The Same method with different implementations in Parent class and child classes.(overriding)

Related to polymorphism the following 4 topics are important

1. **Duck Typing Philosophy of Python**
2. **Overloading**
   1. **Operator Overloading**
   2. **Method Overloading**
   3. **Constructor Overloading**
3. **Overriding**
   1. **Method overriding**
   2. **constructor overriding**

**1. Duck Typing Philosophy of Python:**

In Python we cannot specify the type explicitly. Based on provided value at runtime the type will be considered automatically. Hence Python is considered as Dynamically Typed Programming Language.

def f1(obj): obj.talk()

What is the type of obj? We cannot decide at the beginning. At runtime we can pass any type.Then how we can decide the type?

At runtime if 'it walks like a duck and talks like a duck,it must be duck'. Python follows this principle. This is called Duck Typing Philosophy of Python.



**Demo Program:**

|  |
| --- |
| **1) class Duck:** |
| **2) def talk(self):** |
| **3) print('Quack.. Quack..')** |
| **4)** |
| **5) class Dog:** |
| **6) def talk(self):** |
| **7) print('Bow Bow..')** |
| **8)** |
| **9) class Cat:** |
| **10) def talk(self):** |
| **11) print('Moew Moew ..')** |
| **12)** |
| **13) class Goat:** |
| **14) def talk(self):** |
| **15) print('Myaah Myaah ..')** |
| **16)** |
| **17) def f1(obj):** |
| **18) obj.talk()** |
| **19)** |
| **20) l=[Duck(),Cat(),Dog(),Goat()]** |
| **21) for obj in l:** |
| 22) **f1(obj)** |

**Output:**

Quack.. Quack. . Moew Moew .. Bow Bow..

Myaah Myaah ..

The problem in this approach is if obj does not contain talk() method then we will get AttributeError

**Eg:**

**1) class Duck:**

**3)**

**print('Quack.. Quack..')**

**5) class Dog:**

**7)**

**print('Bow Bow..')**

**9) obj.talk()**

**11) d=Duck() 13)**

**12) f1(d)**

**10)**

**8) def f1(obj):**

**6) def bark(self):**

**4)**

**2) def talk(self):**



14) d=Dog()

15) f1(d)

**Output:**

D:\durga\_classes>py test.py Quack.. Quack..

Traceback (most recent call last):

File "test.py", line 22, in <module> f1(d)

File "test.py", line 13, in f1 obj.talk()

AttributeError: 'Dog' object has no attribute 'talk'

But we can solve this problem by using hasattr() function. hasattr(obj,'attributename')

attributename can be method name or variable name

**Demo Program with hasattr() function:**

**1) class Duck:**

**3)**

**print('Quack.. Quack..')**

**5) class Human:**

**7)**

**print('Hello Hi...')**

**9) class Dog:**

**11)**

**print('Bow Bow..')**

**13) def f1(obj):**

**15)**

**obj.talk()**

**17)**

**obj.bark()**

**19) d=Duck()**

**21)**

**23) f1(h)**

**25) d=Dog()**

**27) Myaah Myaah Myaah...**

**26) f1(d)**

**24)**

**22) h=Human()**

**20) f1(d)**

**18)**

**16) elif hasattr(obj,'bark'):**

**14) if hasattr(obj,'talk'):**

**12)**

**10) def bark(self):**

**8)**

**6) def talk(self):**

**4)**

**2) def talk(self):**



**Overloading:**

We can use same operator or methods for different purposes.

Eg1: + operator can be used for Arithmetic addition and String concatenation print(10+20)#30

print('durga'+'soft')#durgasoft

Eg2: \* operator can be used for multiplication and string repetition purposes. print(10\*20)#200

print('durga'\*3)#durgadurgadurga

Eg3: We can use deposit() method to deposit cash or cheque or dd deposit(cash)

deposit(cheque) deposit(dd)

There are 3 types of overloading

1. **Operator Overloading**
2. **Method Overloading**
3. **Constructor Overloading**
4. Operator Overloading:

We can use the same operator for multiple purposes, which is nothing but operator overloading. Python supports operator overloading.

Eg1: + operator can be used for Arithmetic addition and String concatenation print(10+20)#30

print('durga'+'soft')#durgasoft

Eg2: \* operator can be used for multiplication and string repetition purposes. print(10\*20)#200

print('durga'\*3)#durgadurgadurga

**Demo program to use + operator for our class objects:**

**1) class Book:**

**3)**

**self.pages=pages**

**5) b1=Book(100)**

**7) print(b1+b2)**

**6) b2=Book(200)**

**4)**

**2) def init (self,pages):**



D:\durga\_classes>py test.py Traceback (most recent call last):

File "test.py", line 7, in <module> print(b1+b2)

TypeError: unsupported operand type(s) for +: 'Book' and 'Book'

We can overload + operator to work with Book objects also. i.e Python supports Operator Overloading.

For every operator Magic Methods are available. To overload any operator we have to override that Method in our class.

Internally + operator is implemented by using add () method.This method is called magic method for + operator. We have to override this method in our class.

**Demo program to overload + operator for our Book class objects:**

|  |
| --- |
| **1) class Book:** |
| **2) def init (self,pages):** |
| **3) self.pages=pages** |
| **4)** |
| **5) def add (self,other):** |
| **6) return self.pages+other.pages** |
| **7)** |
| **8) b1=Book(100)** |
| **9) b2=Book(200)** |
| **10) print('The Total Number of Pages:',b1+b2)** |

**Output: The Total Number of Pages: 300**

The following is the list of operators and corresponding magic methods.

+ ---> object. add (self,other)

- ---> object. sub (self,other)

\* ---> object. mul (self,other)

/ ---> object. div (self,other)

// ---> object. floordiv (self,other)

% ---> object. mod (self,other)

\*\* ---> object. pow (self,other)

+= ---> object. iadd (self,other)

-= ---> object. isub (self,other)

\*= ---> object. imul (self,other)

/= ---> object. idiv (self,other)

//= ---> object. ifloordiv (self,other)

%= ---> object. imod (self,other)

\*\*= ---> object. ipow (self,other)

< ---> object. lt (self,other)

<= ---> object. le (self,other)

> ---> object. gt (self,other)

>= ---> object. ge (self,other)



== ---> object. eq (self,other)

!= ---> object. ne (self,other)

**Overloading > and <= operators for Student class objects:**

**1) class Student:**

**3)**

**self.name=name**

**5) def gt (self,other):**

**7) def le (self,other): 9)**

**11) print("10>20 =",10>20)**

**13) s2=Student("Ravi",200) 15) print("s1<s2=",s1<s2)**

**17) print("s1>=s2=",s1>=s2)**

**16) print("s1<=s2=",s1<=s2)**

**14) print("s1>s2=",s1>s2)**

**12) s1=Student("Durga",100)**

**10)**

**return self.marks<=other.marks**

**8)**

**return self.marks>other.marks**

**6)**

**self.marks=marks**

**4)**

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

**Output:**

10>20 = False s1>s2= False s1<s2= True s1<=s2= True s1>=s2= False

**Program to overload multiplication operator to work on Employee objects:**

**1) class Employee:**

**3)**

**self.name=name**

**5) def mul (self,other):**

**7)**

**9) def init (self,name,days):**

**11)**

**self.days=days**

**13) e=Employee('Durga',500)**

**15) print('This Month Salary:',e\*t)**

**14) t=TimeSheet('Durga',25)**

**12)**

**self.name=name**

**10)**

**8) class TimeSheet:**

**return self.salary\*other.days**

**6)**

**self.salary=salary**

**4)**

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

**Output: This Month Salary: 12500**



1. Method Overloading:

If 2 methods having same name but different type of arguments then those methods are said to be overloaded methods.

Eg: m1(int a) m1(double d)

But in Python Method overloading is not possible.

If we are trying to declare multiple methods with same name and different number of arguments then Python will always consider only last method.

**Demo Program:**

|  |
| --- |
| **1) class Test:** |
| **2) def m1(self):** |
| **3) print('no-arg method')** |
| **4) def m1(self,a):** |
| **5) print('one-arg method')** |
| **6) def m1(self,a,b):** |
| **7) print('two-arg method')** |
| **8)** |
| **9) t=Test()** |
| **10) #t.m1()** |
| **11) #t.m1(10)** |
| **12) t.m1(10,20)** |

**Output: two-arg method**

In the above program python will consider only last method.

**How we can handle overloaded method requirements in Python:**

Most of the times, if method with variable number of arguments required then we can handle with default arguments or with variable number of argument methods.

**Demo Program with Default Arguments:**

|  |
| --- |
| **1) class Test:** |
| **2) def sum(self,a=None,b=None,c=None):** |
| **3) if a!=None and b!= None and c!= None:** |
| **4) print('The Sum of 3 Numbers:',a+b+c)** |
| **5) elif a!=None and b!= None:** |
| **6) print('The Sum of 2 Numbers:',a+b)** |
| **7) else:** |
| **8) print('Please provide 2 or 3 arguments')** |
| **9)** |
| **10) t=Test()** |



11) t.sum(10,20)

12) t.sum(10,20,30)

13) t.sum(10)

**Output:**

The Sum of 2 Numbers: 30

The Sum of 3 Numbers: 60

Please provide 2 or 3 arguments

**Demo Program with Variable Number of Arguments:**

**1) class Test:**

**3)**

**total=0**

**5)**

**total=total+x**

**7)**

**9) t=Test()**

**11) t.sum(10,20,30)**

**13) t.sum()**

**12) t.sum(10)**

**10) t.sum(10,20)**

**8)**

**print('The Sum:',total)**

**6)**

**for x in a:**

**4)**

**2) def sum(self,\*a):**

1. Constructor Overloading:

Constructor overloading is not possible in Python.

If we define multiple constructors then the last constructor will be considered.

|  |
| --- |
| **1) class Test:** |
| **2) def init (self):** |
| **3) print('No-Arg Constructor')** |
| **4)** |
| **5) def init (self,a):** |
| **6) print('One-Arg constructor')** |
| **7)** |
| **8) def init (self,a,b):** |
| **9) print('Two-Arg constructor')** |
| **10) #t1=Test()** |
| **11) #t1=Test(10)** |
| **12) t1=Test(10,20)** |

**Output: Two-Arg constructor**



In the above program only Two-Arg Constructor is available.

But based on our requirement we can declare constructor with default arguments and variable number of arguments.

Constructor with Default Arguments:

|  |
| --- |
| **1) class Test:** |
| **2) def init (self,a=None,b=None,c=None):** |
| **3) print('Constructor with 0|1|2|3 number of arguments')** |
| **4)** |
| **5) t1=Test()** |
| **6) t2=Test(10)** |
| **7) t3=Test(10,20)** |
| **8) t4=Test(10,20,30)** |

**Output:**

Constructor with 0|1|2|3 number of arguments Constructor with 0|1|2|3 number of arguments Constructor with 0|1|2|3 number of arguments Constructor with 0|1|2|3 number of arguments

**Constructor with Variable Number of Arguments:**

**1) class Test:**

**3)**

**print('Constructor with variable number of arguments')**

**5) t1=Test()**

**7) t3=Test(10,20)**

**9) t5=Test(10,20,30,40,50,60)**

**8) t4=Test(10,20,30)**

**6) t2=Test(10)**

**4)**

**2) def init (self,\*a):**

**Output:**

Constructor with variable number of arguments Constructor with variable number of arguments Constructor with variable number of arguments Constructor with variable number of arguments Constructor with variable number of arguments



Method overriding:

What ever members available in the parent class are bydefault available to the child class through inheritance. If the child class not satisfied with parent class implementation then child class is allowed to redefine that method in the child class based on its requirement. This concept is called overriding.

Overriding concept applicable for both methods and constructors.

**Demo Program for Method overriding:**

|  |
| --- |
| **1) class P:** |
| **2) def property(self):** |
| **3) print('Gold+Land+Cash+Power')** |
| **4) def marry(self):** |
| **5) print('Appalamma')** |
| **6) class C(P):** |
| **7) def marry(self):** |
| **8) print('Katrina Kaif')** |
| **9)** |
| **10) c=C()** |
| **11) c.property()** |
| 12) **c.marry()** |

**Output:**

Gold+Land+Cash+Power Katrina Kaif

From Overriding method of child class,we can call parent class method also by using super() method.

**1) class P:**

**3)**

**print('Gold+Land+Cash+Power')**

**5)**

**print('Appalamma')**

**7) def marry(self):**

**9)**

**print('Katrina Kaif')**

**11) c=C()**

**13) c.marry()**

**12) c.property()**

**10)**

**super().marry()**

**8)**

**6) class C(P):**

**4) def marry(self):**

**2) def property(self):**

**Output: Gold+Land+Cash+Power Appalamma**

Katrina Kaif



**Demo Program for Constructor overriding:**

**1) class P:**

**3)**

**print('Parent Constructor')**

**5) class C(P):**

**7)**

**print('Child Constructor')**

**9) c=C()**

**8)**

**6) def init (self):**

**4)**

**2) def init (self):**

**Output: Child Constructor**

In the above example,if child class does not contain constructor then parent class constructor will be executed

From child class constuctor we can call parent class constructor by using super() method.

**Demo Program to call Parent class constructor by using super():**

**1) class Person:**

**3)**

**self.name=name**

**5)**

**7) def init (self,name,age,eno,esal):**

**9)**

**self.eno=eno**

**11)**

**13)**

**print('Employee Name:',self.name)**

**15)**

**print('Employee Number:',self.eno)**

**17)**

**19) e1.display()**

**21) e2.display()**

**20) e2=Employee('Sunny',39,872426,36000)**

**18) e1=Employee('Durga',48,872425,26000)**

**print('Employee Salary:',self.esal)**

**16)**

**print('Employee Age:',self.age)**

**14)**

**12) def display(self):**

**self.esal=esal**

**10)**

**super(). init (name,age)**

**8)**

**6) class Employee(Person):**

**self.age=age**

**4)**

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

**Output:**

Employee Name: Durga Employee Age: 48

Employee Number: 872425

Employee Salary: 26000 Employee Name: Sunny Employee Age: 39



Employee Number: 872426

Employee Salary: 36000



**Regular Expressions**

If we want to represent a group of Strings according to a particular format/pattern then we should go for Regular Expressions.

i.e Regualr Expressions is a declarative mechanism to represent a group of Strings accroding to particular format/pattern.

Eg 1: We can write a regular expression to represent all mobile numbers

Eg 2: We can write a regular expression to represent all mail ids.

The main important application areas of Regular Expressions are

1. **To develop validation frameworks/validation logic**
2. **To develop Pattern matching applications (ctrl-f in windows, grep in UNIX etc)**
3. **To develop Translators like compilers, interpreters etc**
4. **To develop digital circuits**
5. **To develop communication protocols like TCP/IP, UDP etc.**

We can develop Regular Expression Based applications by using python module: re

This module contains several inbuilt functions to use Regular Expressions very easily in our applications.

1. **compile()**

re module contains compile() function to compile a pattern into RegexObject.

pattern = re.compile("ab")

1. **finditer():**

Returns an Iterator object which yields Match object for every Match

matcher = pattern.finditer("abaababa")

On Match object we can call the following methods.

1. **start()**  **Returns start index of the match**
2. **end()**  **Returns end+1 index of the match**
3. **group()**  **Returns the matched string**



**Eg:**

|  |
| --- |
| **1) import re count=0** |
| **2) pattern=re.compile("ab")** |
| **3) matcher=pattern.finditer("abaababa")** |
| **4) for match in matcher:** |
| **5) count+=1** |
| **6) print(match.start(),"...",match.end(),"...",match.group())** |
| **7) print("The number of occurrences: ",count)** |

**Output:**

0 ... 2 ... ab

3 ... 5 ... ab

5 ... 7 ... ab

The number of occurrences: 3

Note: We can pass pattern directly as argument to finditer() function.

**Eg:**

* 1. **import re**

2) count=0

3) matcher=re.finditer("ab","abaababa")

4) for match in matcher:

5) count+=1

6) print(match.start(),"...",match.end(),"...",match.group())

7) print("The number of occurrences: ",count)

**Output:**

0 ... 2 ... ab

3 ... 5 ... ab

5 ... 7 ... ab

The number of occurrences: 3

**Character classes:**

We can use character classes to search a group of characters

1. **[abc]===>Either a or b or c**
2. **[^abc] ===>Except a and b and c**
3. **[a-z]==>Any Lower case alphabet symbol**
4. **[A-Z]===>Any upper case alphabet symbol**
5. **[a-zA-Z]==>Any alphabet symbol**
6. **[0-9] Any digit from 0 to 9**
7. **[a-zA-Z0-9]==>Any alphanumeric character**
8. **[^a-zA-Z0-9]==>Except alphanumeric characters(Special Characters)**



**Eg:**

|  |
| --- |
| **1) import re** |
| **2) matcher=re.finditer("x","a7b@k9z")** |
| **3) for match in matcher:** |
| **4) print(match.start()," ",match.group())** |

**x = [abc]**

0 a 2 b

**x = [^abc]**

1 7

3 ...... @

4 k

5 ...... 9

6 z

**x = [a-z]**

0 a 2 b

4 k

6 z

**x = [0-9]**

1 ...... 7

5 ...... 9

**x = [a-zA-Z0-9]**

0 a

1 ...... 7

2 b

4 k

5 ...... 9

6 z

**x = [^a-zA-Z0-9]**

3 ...... @

**Pre defined Character classes:**

\s  Space character

\S  Any character except space character

\d  Any digit from 0 to 9

\D  Any character except digit

\w  Any word character [a-zA-Z0-9]

\W  Any character except word character (Special Characters)

.  Any character including special characters



**Eg:**

|  |
| --- |
| **1) import re** |
| **2) matcher=re.finditer("x","a7b k@9z")** |
| **3) for match in matcher:** |
| **4) print(match.start()," ",match.group())** |

**x = \s:**

3 ......

**x = \S:**

0 a 1 7

2 b

4 k

5 ...... @

6 ...... 9

7 z

**x = \d:**

1 7

6 ...... 9

**x = \D:**

0 a 2 b

3 ......

4 k

5 ...... @

7 z

**x = \w:**

0 a 1 7

2 b

4 k

6 ...... 9

7 z

**x = \W:**

3 ......

5 ...... @

**x = .**

0 a

1 ...... 7

2 b

3 ......



4 k

5 ...... @

6 ...... 9

7 z

Qunatifiers:

We can use quantifiers to specify the number of occurrences to match.

a  Exactly one 'a' a+  Atleast one 'a'

a\*  Any number of a's including zero number

a?  Atmost one 'a' ie either zero number or one number a{m}  Exactly m number of a's

a{m,n}  Minimum m number of a's and Maximum n number of a's

**Eg:**

|  |
| --- |
| **1) import re** |
| **2) matcher=re.finditer("x","abaabaaab")** |
| **3) for match in matcher:** |
| **4) print(match.start()," ",match.group())** |

**x = a:**

0 a

2 a

3 a

5 a

6 a

7 a

**x = a+:**

0 a 2 aa

5 aaa

**x = a\*:**

0 a 1 ......

2 aa

4 ......

5 aaa

8 ......

9 ......



**x = a?:**

0 a

1 ......

2 a

3 a

4 ......

5 a

6 a

7 a

8 ......

9 ......

**x = a{3}:**

5 aaa

**x = a{2,4}:**

2 aa

5 aaa

**Note:**

^x  It will check whether target string starts with x or not x$  It will check whether target string ends with x or not

**Important functions of re module:**

1. **match()**
2. **fullmatch()**
3. **search() 4.findall() 5.finditer()**

6. sub() 7.subn()

1. **split()**
2. **compile()**
3. **match():**

We can use match function to check the given pattern at beginning of target string. If the match is available then we will get Match object, otherwise we will get None.

**Eg:**

|  |
| --- |
| **1) import re** |
| **2) s=input("Enter pattern to check: ")** |
| **3) m=re.match(s,"abcabdefg")** |
| **4) if m!= None:** |
| **5) print("Match is available at the beginning of the String")** |
| **6) print("Start Index:",m.start(), "and End Index:",m.end())** |



|  |
| --- |
| **7) else:** |
| **8) print("Match is not available at the beginning of the String")** |

**Output:**

D:\python\_classes>py test.py Enter pattern to check: abc

Match is available at the beginning of the String Start Index: 0 and End Index: 3

D:\python\_classes>py test.py Enter pattern to check: bde

Match is not available at the beginning of the String

1. **fullmatch():**

We can use fullmatch() function to match a pattern to all of target string. i.e complete string should be matched according to given pattern.

If complete string matched then this function returns Match object otherwise it returns None.

**Eg:**

* 1. **import re**

2) s=input("Enter pattern to check: ")

3) m=re.fullmatch(s,"ababab")

4) if m!= None:

5) print("Full String Matched")

6) else:

7) print("Full String not Matched")

**Output:**

D:\python\_classes>py test.py Enter pattern to check: ab Full String not Matched

D:\python\_classes>py test.py Enter pattern to check: ababab Full String Matched

1. **search():**

We can use search() function to search the given pattern in the target string.

If the match is available then it returns the Match object which represents first occurrence of the match.

If the match is not available then it returns None



**Eg:**

|  |
| --- |
| **1) import re** |
| **2) s=input("Enter pattern to check: ")** |
| **3) m=re.search(s,"abaaaba")** |
| **4) if m!= None:** |
| **5) print("Match is available")** |
| **6) print("First Occurrence of match with start index:",m.start(),"and end index:",m.end())** |
| **7) else:** |
| **8) print("Match is not available")** |

**Output:**

D:\python\_classes>py test.py Enter pattern to check: aaa Match is available

First Occurrence of match with start index: 2 and end index: 5

D:\python\_classes>py test.py Enter pattern to check: bbb Match is not available

1. **findall():**

To find all occurrences of the match.

This function returns a list object which contains all occurrences.

**Eg:**

* 1. **import re**

2) l=re.findall("[0-9]","a7b9c5kz")

3) **print(l)**

**Output: ['7', '9', '5']**

1. **finditer():**

Returns the iterator yielding a match object for each match.

On each match object we can call start(), end() and group() functions.

**Eg:**

|  |
| --- |
| **1) import re** |
| **2) itr=re.finditer("[a-z]","a7b9c5k8z")** |
| **3) for m in itr:** |
| **4) print(m.start(),"...",m.end(),"...",m.group())** |



Output: D:\python\_classes>py test.py 0 ... 1 ... a

2 ... 3 ... b

4 ... 5 ... c

6 ... 7 ... k

8 ... 9 ... z

1. **sub():**

sub means substitution or replacement

re.sub(regex,replacement,targetstring)

In the target string every matched pattern will be replaced with provided replacement.

**Eg:**

* 1. **import re**

2) s=re.sub("[a-z]","#","a7b9c5k8z")

3) print(s)

**Output: #7#9#5#8#**

Every alphabet symbol is replaced with # symbol

1. **subn():**

It is exactly same as sub except it can also returns the number of replacements.

This function returns a tuple where first element is result string and second element is number of replacements.

(resultstring, number of replacements)

**Eg:**

* 1. **import re**

2) t=re.subn("[a-z]","#","a7b9c5k8z")

3) print(t)

4) print("The Result String:",t[0])

5) print("The number of replacements:",t[1])

Output: D:\python\_classes>py test.py ('#7#9#5#8#', 5)

The Result String: #7#9#5#8# The number of replacements: 5



1. **split():**

If we want to split the given target string according to a particular pattern then we should go for split() function.

This function returns list of all tokens.

**Eg:**

* 1. **import re**

2) l=re.split(",","sunny,bunny,chinny,vinny,pinny")

3) print(l)

4) for t in l:

5) print(t)

**Output:**

D:\python\_classes>py test.py

['sunny', 'bunny', 'chinny', 'vinny', 'pinny'] sunny

bunny chinny vinny pinny

**Eg:**

|  |
| --- |
| **1) import re** |
| **2) l=re.split("\.","**[**www.durgasoft.com"**](http://www.durgasoft.com/)**)** |
| **3) for t in l:** |
| **4) print(t)** |

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

durgasoft com

**^ symbol:**

We can use ^ symbol to check whether the given target string starts with our provided pattern or not.

**Eg:**

res=re.search("^Learn",s)

if the target string starts with Learn then it will return Match object,otherwise returns None.



**test.py:**

1) import re

2) s="Learning Python is Very Easy"

3) res=re.search("^Learn",s)

4) if res != None:

5) print("Target String starts with Learn")

6) else:

7) print("Target String Not starts with Learn")

**Output: Target String starts with Learn**

**$ symbol:**

We can use $ symbol to check whether the given target string ends with our provided pattern or not

Eg: res=re.search("Easy$",s)

If the target string ends with Easy then it will return Match object,otherwise returns None.

**test.py:**

1) import re

2) s="Learning Python is Very Easy"

3) res=re.search("Easy$",s)

4) if res != None:

5) print("Target String ends with Easy")

6) else:

7) print("Target String Not ends with Easy")

**Output: Target String ends with Easy**

Note: If we want to ignore case then we have to pass 3rd argument re.IGNORECASE for search() function.

Eg: res = re.search("easy$",s,re.IGNORECASE)

**test.py:**

1) import re

2) s="Learning Python is Very Easy"

3) res=re.search("easy$",s,re.IGNORECASE)

4) if res != None:

5) print("Target String ends with Easy by ignoring case")

6) else:

7) print("Target String Not ends with Easy by ignoring case")



Output: Target String ends with Easy by ignoring case

**App1: Write a Regular Expression to represent all Yava language identifiers**

**Rules:**

1. **The allowed characters are a-z,A-Z,0-9,#**
2. **The first character should be a lower case alphabet symbol from a to k**
3. **The second character should be a digit divisible by 3**
4. **The length of identifier should be atleast 2. [a-k][0369][a-zA-Z0-9#]\***

**App2: Write a python program to check whether the given string is Yava language identifier or not?**

* 1. **import re**

2) s=input("Enter Identifier:")

3) m=re.fullmatch("[a-k][0369][a-zA-Z0-9#]\*",s)

4) if m!= None:

5) print(s,"is valid Yava Identifier")

6) else:

7) print(s,"is invalid Yava Identifier")

**Output:**

D:\python\_classes>py test.py Enter Identifier:a6kk9z## a6kk9z## is valid Yava Identifier

D:\python\_classes>py test.py Enter Identifier:k9b876 k9b876 is valid Yava Identifier

D:\python\_classes>py test.py Enter Identifier:k7b9

k7b9 is invalid Yava Identifier

**App3: Write a Regular Expression to represent all 10 digit mobile numbers.**

**Rules:**

1. **Every number should contains exactly 10 digits**
2. **The first digit should be 7 or 8 or 9**

[7-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]

or



[7-9][0-9]{9}

or

[7-9]\d{9}

**App4: Write a Python Program to check whether the given number is valid mobile number or not?**

* 1. **import re**

2) n=input("Enter number:")

3) m=re.fullmatch("[7-9]\d{9}",n)

4) if m!= None:

5) print("Valid Mobile Number")

6) else:

7) print("Invalid Mobile Number")

**Output:**

D:\python\_classes>py test.py Enter number:9898989898 Valid Mobile Number

D:\python\_classes>py test.py Enter number:6786786787 Invalid Mobile Number

D:\python\_classes>py test.py Enter number:898989

Invalid Mobile Number

**App5: Write a python program to extract all mobile numbers present in input.txt where numbers are mixed with normal text data**

|  |
| --- |
| **1) import re** |
| **2) f1=open("input.txt","r")** |
| **3) f2=open("output.txt","w")** |
| **4) for line in f1:** |
| **5) list=re.findall("[7-9]\d{9}",line)** |
| **6) for n in list:** |
| **7) f2.write(n+"\n")** |
| **8) print("Extracted all Mobile Numbers into output.txt")** |
| **9) f1.close()** |
| **10) f2.close()** |



Web Scraping by using Regular Expressions:

The process of collecting information from web pages is called web scraping. In web scraping to match our required patterns like mail ids, mobile numbers we can use regular expressions.

**Eg:**

|  |
| --- |
| **1) import re,urllib** |
| **2) import urllib.request** |
| **3) sites="google rediff".split()** |
| **4) print(sites)** |
| **5) for s in sites:** |
| **6) print("Searching...",s)** |
| **7) u=urllib.request.urlopen("http://"+s+".com")** |
| **8) text=u.read()** |
| **9) title=re.findall("<title>.\*</title>",str(text),re.I)** |
| 10) **print(title[0])** |

**Eg: Program to get all phone numbers of redbus.in by using web scraping and regular expressions**

1) import re,urllib

2) import urllib.request

3) u=urllib.request.urlopen("https://[www.redbus.in/info/contactus](http://www.redbus.in/info/contactus)")

4) text=u.read()

5) numbers=re.findall("[0-9-]{7}[0-9-]+",str(text),re.I)

6) for n in numbers:

7) print(n)

**Q. Write a Python Program to check whether the given mail id is valid gmail id or not?**

1) import re

2) s=input("Enter Mail id:")

3) m=re.fullmatch("\w[a-zA-Z0-9\_.]\*@gmail[.]com",s)

4) if m!=None:

5) print("Valid Mail Id");

6) else:

7) print("Invalid Mail id")

**Output:**

D:\python\_classes>py test.py Enter Mail [id:durgatoc@gmail.com](mailto:durgatoc@gmail.com) Valid Mail Id



D:\python\_classes>py test.py Enter Mail id:durgatoc Invalid Mail id

D:\python\_classes>py test.py

Enter Mail [id:durgatoc@yahoo.co.in](mailto:durgatoc@yahoo.co.in) Invalid Mail id

**Q. Write a python program to check whether given car registration number is valid Telangana State Registration number or not?**

1) import re

2) s=input("Enter Vehicle Registration Number:")

3) m=re.fullmatch("TS[012][0-9][A-Z]{2}\d{4}",s)

4) if m!=None:

5) print("Valid Vehicle Registration Number");

6) else:

7) print("Invalid Vehicle Registration Number")

**Output:**

D:\python\_classes>py test.py

Enter Vehicle Registration Number:TS07EA7777 Valid Vehicle Registration Number

D:\python\_classes>py test.py

Enter Vehicle Registration Number:TS07KF0786 Valid Vehicle Registration Number

D:\python\_classes>py test.py

Enter Vehicle Registration Number:AP07EA7898 Invalid Vehicle Registration Number

**Q. Python Program to check whether the given mobile number is valid OR not (10 digit OR 11 digit OR 12 digit)**

1) import re

2) s=input("Enter Mobile Number:")

3) m=re.fullmatch("(0|91)?[7-9][0-9]{9}",s)

4) if m!=None:

5) print("Valid Mobile Number");

6) else:

7) print("Invalid Mobile Number")

Summary table and some more examples.





**Python Database Programming**

**Storage Areas**

As the Part of our Applications, we required to store our Data like Customers Information, Billing Information, Calls Information etc..

To store this Data, we required Storage Areas. There are 2 types of Storage Areas.

1. **Temporary Storage Areas**
2. **Permanent Storage Areas**
3. **Temporary Storage Areas:**

These are the Memory Areas where Data will be stored temporarily.

Eg: Python objects like List, Tuple, Dictionary.

Once Python program completes its execution then these objects will be destroyed automatically and data will be lost.

1. **Permanent Storage Areas:**

Also known as Persistent Storage Areas. Here we can store Data permanently.

Eg: File Systems, Databases, Data warehouses, Big Data Technologies etc

**File Systems:**

File Systems can be provided by Local operating System. File Systems are best suitable to store very less Amount of Information.

**Limitations:**

1. **We cannot store huge Amount of Information.**
2. **There is no Query Language support and hence operations will become very complex.**
3. **There is no Security for Data.**
4. **There is no Mechanism to prevent duplicate Data. Hence there may be a chance of Data Inconsistency Problems.**

To overcome the above Problems of File Systems, we should go for Databases.

**Databases:**

1. **We can store Huge Amount of Information in the Databases.**



1. **Query Language Support is available for every Database and hence we can perform Database Operations very easily.**
2. **To access Data present in the Database, compulsory username and pwd must be required. Hence Data is secured.**
3. **Inside Database Data will be stored in the form of Tables. While developing Database Table Schemas, Database Admin follow various Normalization Techniques and can implement various Constraints like Unique Key Constrains, Primary Key Constraints etc which prevent Data Duplication. Hence there is no chance of Data Inconsistency Problems.**

**Limitations of Databases:**

1. **Database cannot hold very Huge Amount of Information like Terabytes of Data.**
2. **Database can provide support only for Structured Data (Tabular Data OR Relational Data) and cannot provide support for Semi Structured Data (like XML Files) and Unstructured Data (like Video Files, Audio Files, Images etc)**

To overcome these Problems we should go for more Advanced Storage Areas like Big Data Technologies, Data warehouses etc.

**Python Database Programming:**

Sometimes as the part of Programming requirement we have to connect to the database and we have to perform several operations like creating tables, inserting data,updating data,deleting data,selecting data etc.

We can use SQL Language to talk to the database and we can use Python to send those SQL commands to the database.

Python provides inbuilt support for several databases like Oracle, MySql, SqlServer, GadFly, sqlite, etc.

Python has seperate module for each database.

Eg: cx\_Oralce module for communicating with Oracle database pymssql module for communicating with Microsoft Sql Server

**Standard Steps for Python database Programming:**

1. **Import database specific module**

Eg: import cx\_Oracle

1. **Establish Connection between Python Program and database.**

We can create this Connection object by using connect() function of the module. con = cx\_Oracle.connect(datbase information)

Eg: con=cx\_Oracle.connect('scott/tiger@localhost')

1. **To execute our sql queries and to hold results some special object is required, which is nothing but Cursor object. We can create Cursor object by using cursor() method.**



cursor=con.cursor()

1. **Execute SQL Queries By using Cursor object. For this we can use the following methods**
2. **execute(sqlquery)**  **To execute a single sql query**
3. **executescript(sqlqueries)**  **To execute a string of sql queries seperated by semi-colon ';'**
4. **executemany()**  **To execute a Parameterized query**

Eg: cursor.execute("select \* from employees")

1. **commit or rollback changes based on our requirement in the case of DML Queries(insert|update|delete)**

commit()  Saves the changes to the database rollback()  rolls all temporary changes back

1. **Fetch the result from the Cursor object in the case of select queries fetchone()**  **To fetch only one row**

fetchall()  To fetch all rows and it returns a list of rows fecthmany(n)  To fetch first n rows

Eg 1: data =cursor.fetchone() print(data)

Eg 2: data=cursor.fetchall() for row in data:

print(row)

1. **close the resources**

After completing our operations it is highly recommended to close the resources in the reverse order of their opening by using close() methods.

cursor.close() con.close()

Note: The following is the list of all important methods which can be used for python database programming.

connect() cursor() execute() executescript() executemany() commit() rollback() fetchone() fetchall() fetchmany(n)



fetch close()

These methods won't be changed from database to database and same for all databases.

**Working with Oracle Database:**

From Python Program if we want to communicate with any database,some translator must be required to translate Python calls into Database specific calls and Database specific calls into Python calls.This translator is nothing but Driver/Connector.

Diagram

For Oracle database the name of driver needed is cx\_Oracle.

cx\_Oracle is a Python extension module that enables access to Oracle Database.It can be used for both Python2 and Python3. It can work with any version of Oracle database like 9,10,11 and 12.

**Installing cx\_Oracle:**

From Normal Command Prompt (But not from Python console)execute the following command D:\python\_classes>pip install cx\_Oracle

Collecting cx\_Oracle

Downloading cx\_Oracle-6.0.2-cp36-cp36m-win32.whl (100kB) 100% | | 102kB 256kB/s

Installing collected packages: cx-Oracle Successfully installed cx-Oracle-6.0.2

**How to Test Installation:**

From python console execute the following command:

>>> help("modules")

In the output we can see cx\_Oracle

....

\_multiprocessing crypt ntpath timeit

\_opcode csv nturl2path tkinter

\_operator csvr numbers token

\_osx\_support csvw opcode tokenize

\_overlapped ctypes operator trace

\_pickle curses optparse traceback

\_pydecimal custexcept os tracemalloc

\_pyio cx\_Oracle parser try

\_random data pathlib tty

\_sha1 datetime pdb turtle



|  |  |  |  |
| --- | --- | --- | --- |
| **\_sha256** | **dbm** | **pick** | **turtledemo** |
| **\_sha3** | **decimal** | **pickle** | **types** |

\_sha512 demo pickletools typing

\_signal difflib pip unicodedata

\_sitebuiltins dis pipes unittest

\_socket distutils pkg\_resources unpick

\_sqlite3 doctest pkgutil update

\_sre dummy\_threading platform urllib

\_ssl durgamath plistlib uu

\_stat easy\_install polymorph uuid

.....

**App1: Program to connect with Oracle database and print its version.**

* 1. **import** cx\_Oracle

2) con=cx\_Oracle.connect('scott/tiger@localhost')

3) **print**(con.version)

4) con.close()

**Output:**

D:\python\_classes>py db1.py 11.2.0.2.0

**App2: Write a Program to create employees table in the oracle database :**

employees(eno,ename,esal,eaddr)

1) **import** cx\_Oracle

3)

con=cx\_Oracle.connect('scott/tiger@localhost')

5)

cursor.execute("create table employees(eno number,ename varchar2(10),esal number(10,2),eaddr varchar2(10))")

7) **except** cx\_Oracle.DatabaseError as e:

9) con.rollback()

11) **finally**:

13) cursor.close()

15)

con.close()

**if** con:

14)

**if** cursor:

12)

**print**("There is a problem with sql",e)

10)

**if** con:

8)

**print**("Table created successfully")

6)

cursor=con.cursor()

4)

2) **try**:

**App3: Write a program to drop employees table from oracle database?**

1) **import** cx\_Oracle

3)

con=cx\_Oracle.connect('scott/tiger@localhost')

5) cursor.execute("drop table employees")

7) **except** cx\_Oracle.DatabaseError as e:

9) con.rollback()

11) **finally**:

13) cursor.close()

15)

con.close()

**if** con:

14)

**if** cursor:

12)

**print**("There is a problem with sql",e)

10)

**if** con:

8)

**print**("Table dropped successfully")

6)

cursor=con.cursor()

4)

2) **try**:

**App3: Write a program to insert a single row in the employees table.**



1) **import** cx\_Oracle

3)

con=cx\_Oracle.connect('scott/tiger@localhost')

5) cursor.execute("insert into employees values(100,'Durga',1000,'Hyd')")

7) **print**("Record Inserted Successfully")

9) **if** con:

11) **print**("There is a problem with sql",e)

13) **if** cursor:

15) **if** con:

con.close()

16)

cursor.close()

14)

12) **finally**:

con.rollback()

10)

8) **except** cx\_Oracle.DatabaseError as e:

con.commit()

6)

cursor=con.cursor()

4)

2) **try**:

Note: While performing DML Operations (insert|update|delte), compulsory we have to use commit() method,then only the results will be reflected in the database.

**App4: Write a program to insert multiple rows in the employees table by using executemany() method.**

1) **import** cx\_Oracle

3)

con=cx\_Oracle.connect('scott/tiger@localhost')

5) sql="insert into employees values(:eno,:ename,:esal,:eaddr)" 7) (300,'Chinny',3000,'Hyd'),

9) cursor.executemany(sql,records)

11) **print**("Records Inserted Successfully")

13) **if** con:

15) **print**("There is a problem with sql",e)

17) **if** cursor:

19) **if** con:

con.close()

20)

cursor.close()

18)

16) **finally**:

con.rollback()

14)

12) **except** cx\_Oracle.DatabaseError as e:

con.commit()

10)

(400,'Bunny',4000,'Hyd')]

8)

records=[(200,'Sunny',2000,'Mumbai'),

6)

cursor=con.cursor()

4)

2) **try**:

**App5: Write a program to insert multiple rows in the employees table with dynamic input from the keyboard?**

1) **import** cx\_Oracle

3)

con=cx\_Oracle.connect('scott/tiger@localhost')

5) **while** True:

7)

ename=input("Enter Employee Name:")

9) eaddr=input("Enter Employee Address:")

11) cursor.execute(sql %(eno,ename,esal,eaddr))

13) option=input("Do you want to insert one more record[Yes|No] :")

15) con.commit()

17) **except** cx\_Oracle.DatabaseError as e:

19) con.rollback()

21) **finally**:

23) cursor.close()

25)

con.close()

**if** con:

24)

**if** cursor:

22)

**print**("There is a problem with sql :",e)

20)

**if** con:

18)

**break**

16)

**if** option=="No":

14)

**print**("Record Inserted Successfully")

12)

sql="insert into employees values(%d,'%s',%f,'%s')"

10)

esal=float(input("Enter Employee Salary:"))

8)

eno=int(input("Enter Employee Number:"))

6)

cursor=con.cursor()

4)

2) **try**:



**App6: Write a program to update employee salaries with increment for the certain range with dynamic input.**

Eg: Increment all employee salaries by 500 whose salary < 5000

1) **import** cx\_Oracle

3)

con=cx\_Oracle.connect('scott/tiger@localhost')

5) increment=float(input("Enter Increment Salary:"))

7) sql="update employees set esal=esal+%f where esal<%f"

9) **print**("Records Updated Successfully")

11) **except** cx\_Oracle.DatabaseError as e:

13) con.rollback()

15) **finally**:

17) cursor.close()

19)

con.close()

**if** con:

18)

**if** cursor:

16)

**print**("There is a problem with sql :",e)

14)

**if** con:

12)

con.commit()

10)

cursor.execute(sql %(increment,salrange))

8)

salrange=float(input("Enter Salary Range:"))

6)

cursor=con.cursor()

4)

2) **try**:

**App7: Write a program to delete employees whose salary greater provided salary as dynamic input?**

Eg: delete all employees whose salary > 5000

1) **import** cx\_Oracle

3)

con=cx\_Oracle.connect('scott/tiger@localhost')

5) cutoffsalary=float(input("Enter CutOff Salary:"))

7) cursor.execute(sql %(cutoffsalary))

9) con.commit()

11) **if** con:

13) **print**("There is a problem with sql :",e)

15) **if** cursor:

17) **if** con:

con.close()

18)

cursor.close()

16)

14) **finally**:

con.rollback()

12)

10) **except** cx\_Oracle.DatabaseError as e:

**print**("Records Deleted Successfully")

8)

sql="delete from employees where esal>%f"

6)

cursor=con.cursor()

4)

2) **try**:

**App8: Write a program to select all employees info by using fetchone() method?**

1) **import** cx\_Oracle

3)

con=cx\_Oracle.connect('scott/tiger@localhost')

5) cursor.execute("select \* from employees")

7) **while** row **is not** None:

9) row=cursor.fetchone()

11) **if** con:

13) **print**("There is a problem with sql :",e)

15) **if** cursor:

17) **if** con:

con.close()

18)

cursor.close()

16)

14) **finally**:

con.rollback()

12)

10) **except** cx\_Oracle.DatabaseError as e:

**print**(row)

8)

row=cursor.fetchone()

6)

cursor=con.cursor()

4)

2) **try**:



**App9: Write a program to select all employees info by using fetchall() method?**

1) **import** cx\_Oracle

3)

con=cx\_Oracle.connect('scott/tiger@localhost')

5) cursor.execute("select \* from employees")

7) **for** row **in** data:

9) **print**("Employee Name:",row[1])

11) **print**("Employee Address:",row[3])

13) **print**()

15) **if** con:

17) **print**("There is a problem with sql :",e)

19) **if** cursor:

21) **if** con:

con.close()

22)

cursor.close()

20)

18) **finally**:

con.rollback()

16)

14) **except** cx\_Oracle.DatabaseError as e:

**print**()

12)

**print**("Employee Salary:",row[2])

10)

**print**("Employee Number:",row[0])

8)

data=cursor.fetchall()

6)

cursor=con.cursor()

4)

2) **try**:

**App10: Write a program to select employees info by using fetchmany() method and the required number of rows will be provided as dynamic input?**

1) **import** cx\_Oracle

3)

con=cx\_Oracle.connect('scott/tiger@localhost')

5) cursor.execute("select \* from employees")

7) data=cursor.fetchmany(n)

9) **print**(row)

11) **if** con:

13) **print**("There is a problem with sql :",e)

15) **if** cursor:

17) **if** con:

con.close()

18)

cursor.close()

16)

14) **finally**:

con.rollback()

12)

10) **except** cx\_Oracle.DatabaseError as e:

**for** row **in** data:

8)

n=int(input("Enter the number of required rows:"))

6)

cursor=con.cursor()

4)

2) **try**:

**Output:**

D:\python\_classes>py test.py

Enter the number of required rows:3 (100, 'Durga', 1500.0, 'Hyd')

(200, 'Sunny', 2500.0, 'Mumbai')

(300, 'Chinny', 3500.0, 'Hyd')

D:\python\_classes>py test.py

Enter the number of required rows:4 (100, 'Durga', 1500.0, 'Hyd')

(200, 'Sunny', 2500.0, 'Mumbai')

(300, 'Chinny', 3500.0, 'Hyd')

(400, 'Bunny', 4500.0, 'Hyd')

**Working with Mysql database:**

Current version: 5.7.19

Vendor: SUN Micro Systems/Oracle Corporation



Open Source and Freeware Default Port: 3306

Default user: root

Note: In MySQL, everything we have to work with our own databases, which are also known as Logical Databases.

The following are 4 default databases available in mysql.

1. **information\_schema**
2. **mysql**
3. **performance\_schema**
4. **test Diagram**

In the above diagram only one physical database is available and 4 logical databases are available.

**Commonly used commands in MySql:**

1. **To know available databases:**

mysql> show databases;

1. **To create our own logical database**

mysql> create database durgadb;

1. **To drop our own database:**

mysql> drop database durgadb;

1. **To use a particular logical database**

mysql> use durgadb; OR mysql> connect durgadb;

1. **To create a table:**

create table employees(eno int(5) primary key,ename varchar(10),esal double(10,2),eaddr varchar(10));

1. **To insert data:**

insert into employees values(100,'Durga',1000,'Hyd'); insert into employees values(200,'Ravi',2000,'Mumbai');

In MySQL instead of single quotes we can use double quotes also.

Driver/Connector Information:

From Python program if we want to communicates with MySql database,compulsory some translator is required to convert python specific calls into mysql database specific calls and mysql database specific calls into python specific calls. This translator is nothing but Driver or Connector.



Diagram

We have to download connector seperately from mysql database. <https://dev.mysql.com/downloads/connector/python/2.1.html>

**How to check installation:**

From python console we have to use help("modules")

In the list of modules,compulsory mysql should be there.

Note: In the case of Python3.4 we have to set PATH and PYTHONPATH explicitly PATH=C:\Python34

PYTHONPATH=C:\Python34\Lib\site-packages

**Q. Write a Program to create table,insert data and display data by using mysql database.**

1) **import** mysql.connector

3)

con=mysql.connector.connect(host='localhost',database='durgadb',user='root',password='root')

5) cursor.execute("create table employees(eno int(5) primary key,ename varchar(10),esal double(10,2),eaddr varchar( 10))")

7)

9) records=[(100,'Sachin',1000,'Mumbai'),

11) (300,'Kohli',3000,'Delhi')]

13) con.commit() 15)

17) data=cursor.fetchall()

19) **print**("Employee Number:",row[0])

21) **print**("Employee Salary:",row[2])

23) **print**()

25) **except** mysql.connector.DatabaseError as e:

27) con.rollback()

29) **finally**:

31) cursor.close()

33) con.close()

**if** con:

32)

**if** cursor:

30)

**print**("There is a problem with sql :",e)

28)

**if** con:

26)

**print**()

24)

**print**("Employee Address:",row[3])

22)

**print**("Employee Name:",row[1])

20)

**for** row **in** data:

18)

cursor.execute("select \* from employees")

16)

**print**("Records Inserted Successfully...")

14)

cursor.executemany(sql,records)

12)

(200,'Dhoni',2000,'Ranchi'),

10)

sql = "insert into employees(eno, ename, esal, eaddr) VALUES(%s, %s, %s, %s)"

8)

**print**("Table Created...")

6)

cursor=con.cursor()

4)

2) **try**:

**Q. Write a Program to copy data present in employees table of mysql database into Oracle database.**

1) **import** mysql.connector

3) **try**:

con=mysql.connector.connect(host='localhost',database='durgadb',user='root',password='root')

4)

2) **import** cx\_Oracle



5)

cursor=con.cursor()

7)

data=cursor.fetchall()

9)

**for** row **in** data:

11)

list.append(t)

13)

**if** con:

15)

**print**("There is a problem with MySql :",e)

17)

**if** cursor:

19)

**if** con:

21)

23)

con=cx\_Oracle.connect('scott/tiger@localhost')

25)

sql="insert into employees values(:eno,:ename,:esal,:eaddr)"

27)

con.commit()

29) **except** cx\_Oracle.DatabaseError as e:

31)

con.rollback()

33) **finally**:

35) cursor.close()

37)

con.close()

**if** con:

36)

**if** cursor:

34)

**print**("There is a problem with sql",e)

32)

**if** con:

30)

**print**("Records Copied from MySQL Database to Oracle Database Successfully")

28)

cursor.executemany(sql,list)

26)

cursor=con.cursor()

24)

22) **try**:

con.close()

20)

cursor.close()

18)

16) **finally**:

con.rollback()

14)

12) **except** mysql.connector.DatabaseError as e:

t=(row[0],row[1],row[2],row[3])

10)

list=[]

8)

cursor.execute("select \* from employees")

6)

<https://dev.mysql.com/downloads/connector/python/2.1.html>

1) **create table** employees(eno **int**(5) **primary key**,ename **varchar**(10),esal **double**(10,2),eaddr **varchar**(10));

3)

**insert into** employees **values**(100,'Durga',1000,'Hyd');

5)

**insert into** employees **values**(300,'Shiva',3000,'Hyd');

**insert into** employees **values**(200,'Ravi',2000,'Mumbai');

4)

2)



Pattern Programs

Pattern-1:

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

1) n=int(input("Enter the number of rows: "))

2) for i in range(1,n+1):

3) print("\* "\*n)

Pattern-2:

1 1 1 1 1 1 1 1 1 1

2 2 2 2 2 2 2 2 2 2

3 3 3 3 3 3 3 3 3 3

4 4 4 4 4 4 4 4 4 4

5 5 5 5 5 5 5 5 5 5

6 6 6 6 6 6 6 6 6 6

7 7 7 7 7 7 7 7 7 7

8 8 8 8 8 8 8 8 8 8

9 9 9 9 9 9 9 9 9 9

10 10 10 10 10 10 10 10 10 10

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

**3) for j in range(1,n+1):**

**5) print()**

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

**4)**

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

Pattern-3:

1 2 3 4 5 6 7 8 9 10

1 2 3 4 5 6 7 8 9 10

1 2 3 4 5 6 7 8 9 10



1 2 3 4 5 6 7 8 9 10

1 2 3 4 5 6 7 8 9 10

1 2 3 4 5 6 7 8 9 10

1 2 3 4 5 6 7 8 9 10

1 2 3 4 5 6 7 8 9 10

1 2 3 4 5 6 7 8 9 10

1 2 3 4 5 6 7 8 9 10

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

**3) for j in range(1,n+1):**

**5) print()**

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

**4)**

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

Pattern-4:

A A A A A A A A A A B B B B B B B B B B C C C C C C C C C C

D D D D D D D D D D E E E E E E E E E E

F F F F F F F F F F

G G G G G G G G G G H H H H H H H H H H I I I I I I I I I I

J J J J J J J J J J

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

**3) for j in range(1,n+1):**

5)

**print()**

**print(chr(64+i),end=" ")**

**4)**

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

Pattern-5:

A B C D E F G H I J A B C D E F G H I J A B C D E F G H I J A B C D E F G H I J A B C D E F G H I J A B C D E F G H I J A B C D E F G H I J A B C D E F G H I J A B C D E F G H I J A B C D E F G H I J



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

**3) for j in range(1,n+1):**

**5) print()**

**print(chr(64+j),end=" ")**

**4)**

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

Pattern-6:

10 10 10 10 10 10 10 10 10 10

9 9 9 9 9 9 9 9 9 9

8 8 8 8 8 8 8 8 8 8

7 7 7 7 7 7 7 7 7 7

6 6 6 6 6 6 6 6 6 6

5 5 5 5 5 5 5 5 5 5

4 4 4 4 4 4 4 4 4 4

3 3 3 3 3 3 3 3 3 3

2 2 2 2 2 2 2 2 2 2

1 1 1 1 1 1 1 1 1 1

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

**3) for j in range(1,n+1):**

**5) print()**

**print(n+1-i,end=" ")**

**4)**

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

Pattern-6:

10 9 8 7 6 5 4 3 2 1

10 9 8 7 6 5 4 3 2 1

10 9 8 7 6 5 4 3 2 1

10 9 8 7 6 5 4 3 2 1

10 9 8 7 6 5 4 3 2 1

10 9 8 7 6 5 4 3 2 1

10 9 8 7 6 5 4 3 2 1

10 9 8 7 6 5 4 3 2 1

10 9 8 7 6 5 4 3 2 1

10 9 8 7 6 5 4 3 2 1

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

**3) for j in range(1,n+1):**

**5) print()**

**print(n+1-j,end=" ")**

**4)**

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



Pattern-7:

J J J J J J J J J J I I I I I I I I I I

H H H H H H H H H H G G G G G G G G G G F F F F F F F F F F

E E E E E E E E E E

D D D D D D D D D D C C C C C C C C C C

B B B B B B B B B B A A A A A A A A A A

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

**3) for j in range(1,n+1):**

**5) print()**

**print(chr(65+n-i),end=" ")**

**4)**

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

Pattern-8:

J I H G F E D C B A J I H G F E D C B A J I H G F E D C B A J I H G F E D C B A J I H G F E D C B A J I H G F E D C B A J I H G F E D C B A J I H G F E D C B A J I H G F E D C B A J I H G F E D C B A

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

**3) for j in range(1,n+1):**

5)

**print()**

**print(chr(65+n-j),end=" ")**

**4)**

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

Pattern-9:

\*

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

\* \* \* \* \* \*

\* \* \* \* \* \* \*



\* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

**Code - 1:**

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

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

**5) print()**

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

**4)**

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

**Code - 2:**

1) n=int(input("Enter the number of rows:"))

2) for i in range(1,n+1):

3) print("\* "\*i)

Pattern-10:

1

2 2

3 3 3

4 4 4 4

5 5 5 5 5

6 6 6 6 6 6

7 7 7 7 7 7 7

8 8 8 8 8 8 8 8

9 9 9 9 9 9 9 9 9

10 10 10 10 10 10 10 10 10 10

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

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

**5) print()**

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

**4)**

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

Pattern-11:

1

1 2

1 2 3

1 2 3 4

1 2 3 4 5

1 2 3 4 5 6

1 2 3 4 5 6 7

1 2 3 4 5 6 7 8



1 2 3 4 5 6 7 8 9

1 2 3 4 5 6 7 8 9 10

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

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

5)

**print()**

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

**4)**

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

Pattern-12:

A B B

C C C

D D D D E E E E E F F F F F F

G G G G G G G H H H H H H H H I I I I I I I I I

J J J J J J J J J J

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

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

**5) print()**

**print(chr(64+i),end=" ")**

**4)**

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

Pattern-13:

A A B

A B C

A B C D A B C D E

A B C D E F

A B C D E F G

A B C D E F G H A B C D E F G H I

A B C D E F G H I J

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

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

5)

**print()**

**print(chr(64+j),end=" ")**

**4)**

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



Pattern-14:

* **Squares**
* **Right Angled Triangle**
* **Reverse of Right Angled Triangle**

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \*

\* \* \* \* \* \* \*

\* \* \* \* \* \*

\* \* \* \* \*

\* \* \* \*

\* \* \*

\* \*

\*

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

**3) for j in range(1,n+2-i):**

5)

**print()**

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

**4)**

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

Pattern-15:

1 1 1 1 1 1 1 1 1 1

2 2 2 2 2 2 2 2 2

3 3 3 3 3 3 3 3

4 4 4 4 4 4 4

5 5 5 5 5 5

6 6 6 6 6

7 7 7 7

8 8 8

9 9

10

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

**3) for j in range(1,n+2-i):**

**5) print()**

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

**4)**

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



Pattern-16:

1 2 3 4 5 6 7 8 9 10

1 2 3 4 5 6 7 8 9

1 2 3 4 5 6 7 8

1 2 3 4 5 6 7

1 2 3 4 5 6

1 2 3 4 5

1 2 3 4

1 2 3

1 2

1

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

**3) for j in range(1,n+2-i):**

5)

**print()**

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

**4)**

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

Pattern-17:

A A A A A A A A A A B B B B B B B B B

C C C C C C C C D D D D D D D E E E E E E

F F F F F G G G G H H H

I I J

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

**3) for j in range(1,n+2-i):**

**5) print()**

**print(chr(64+i),end=" ")**

**4)**

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

Pattern-18:

A B C D E F G H I J A B C D E F G H I A B C D E F G H

A B C D E F G A B C D E F

A B C D E A B C D



A B C A B A

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

**3) for j in range(1,n+2-i):**

5)

**print()**

**print(chr(64+j),end=" ")**

**4)**

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

Pattern-19:

10 10 10 10 10 10 10 10 10 10

9 9 9 9 9 9 9 9 9

8 8 8 8 8 8 8 8

7 7 7 7 7 7 7

6 6 6 6 6 6

5 5 5 5 5

4 4 4 4

3 3 3

2 2

1

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

**3) for j in range(1,n+2-i):**

**5) print()**

**print(n+1-i,end=" ")**

**4)**

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

Pattern-20:

10 9 8 7 6 5 4 3 2 1

10 9 8 7 6 5 4 3 2

10 9 8 7 6 5 4 3

10 9 8 7 6 5 4

10 9 8 7 6 5

10 9 8 7 6

10 9 8 7

10 9 8

10 9

10

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

**3) for j in range(1,n+2-i):**

**5) print()**

**print(n+1-j,end=" ")**

**4)**

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



Pattern-21:

J J J J J J J J J J I I I I I I I I I

H H H H H H H H G G G G G G G

F F F F F F E E E E E D D D D C C C

B B A

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

**3) for j in range(1,n+2-i):**

**5) print()**

**print(chr(65+n-i),end=" ")**

**4)**

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

Pattern-22:

J I H G F E D C B A J I H G F E D C B

J I H G F E D C J I H G F E D

J I H G F E J I H G F

J I H G J I H

J I J

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

**3) for j in range(1,n+2-i):**

**5) print()**

**print(chr(65+n-j),end=" ")**

**4)**

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

Pattern-23:

\*

\*\*

\*\*\*

\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*\*

\*\*\*\*\*\*\*



\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*

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

Pattern-24:

\*

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

\* \* \* \* \* \*

\* \* \* \* \* \* \*

\* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

|  |
| --- |
| **1) n=int(input("Enter the number of rows: "))** |
| **2) for i in range(1,n+1):** |
| **3) print(" "\*(n-i),end="")** |
| **4) for j in range(1,i+1):** |
| **5) print("\*",end=" ")** |
| **6) print()** |

Pattern-25:

1

2 2

3 3 3

4 4 4 4

5 5 5 5 5

6 6 6 6 6 6

7 7 7 7 7 7 7

8 8 8 8 8 8 8 8



9 9 9 9 9 9 9 9 9

10 10 10 10 10 10 10 10 10 10

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

Pattern-26:

1

1 2

1 2 3

1 2 3 4

1 2 3 4 5

1 2 3 4 5 6

1 2 3 4 5 6 7

1 2 3 4 5 6 7 8

1 2 3 4 5 6 7 8 9

1 2 3 4 5 6 7 8 9 10

|  |
| --- |
| **1) n=int(input("Enter the number of rows: "))** |
| **2) for i in range(1,n+1):** |
| **3) print(" "\*(n-i),end="")** |
| **4) for j in range(1,i+1):** |
| **5) print(j,end=" ")** |
| **6) print()** |

Pattern-27:

A B B C C C

D D D D E E E E E

F F F F F F

G G G G G G G H H H H H H H H



|  |
| --- |
| **1) n=int(input("Enter the number of rows: "))** |
| **2) for i in range(1,n+1):** |
| **3) print(" "\*(n-i),(chr(64+i)+" ")\*i)** |
| **4) print()** |

Pattern-28:

A A B A B C

A B C D A B C D E A B C D E F

A B C D E F G A B C D E F G H A B C D E F G H I

A B C D E F G H I J

1) n=int(input("Enter the number of rows: "))

|  |
| --- |
| **2) for i in range(1,n+1):** |
| **3) print(" "\*(n-i),end="")** |
| **4) for j in range(1,i+1):** |
| **5) print(chr(64+j),end=" ")** |
| **6) print()** |

Pattern-29:

\* \* \* \* \*

\* \* \* \*

\* \* \*

\* \*

\*

1) n=int(input("Enter the number of rows: "))

2) for i in range(1,n+1):

3) print(" "\*(i-1),"\* "\*(n+1-i))

Pattern-30:

5 5 5 5 5

4 4 4 4

3 3 3

2 2

1



1) n=int(input("Enter the number of rows: "))

2) for i in range(1,n+1):

3) print(" "\*(i-1),(str(n+1-i)+" ")\*(n+1-i))

Pattern-31:

1 2 3 4 5

1 2 3 4

1 2 3

1 2

1

|  |
| --- |
| **1) n=int(input("Enter the number of rows: "))** |
| **2) for i in range(1,n+1):** |
| **3) print(" "\*(i-1),end="")** |
| **4) for j in range(1,n+2-i):** |
| **5) print(j,end=" ")** |
| **6) print()** |

Pattern-32:

E E E E E D D D D

C C C B B

A

1) n=int(input("Enter the number of rows: "))

2) for i in range(1,n+1):

3) print(" "\*(i-1),(str(chr(65+n-i))+" ")\*(n+1-i))

Pattern-33:

A B C D E A B C D

A B C A B

A

|  |
| --- |
| **1) n=int(input("Enter the number of rows: "))** |
| **2) for i in range(1,n+1):** |
| **3) print(" "\*(i-1),end="")** |
| **4) for j in range(65,66+n-i):** |
| **5) print(chr(j),end=" ")** |
| **6) print()** |



Pattern-34:

\*

\* \* \*

\* \* \* \* \*

\* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \*

1) n=int(input("Enter the number of rows: "))

2) for i in range(1,n+1):

3) print(" "\*(n-i),"\* "\*(2\*i-1))

Pattern-35:

1

2 2 2

3 3 3 3 3

4 4 4 4 4 4 4

5 5 5 5 5 5 5 5 5

1) n=int(input("Enter the number of rows: "))

2) for i in range(1,n+1):

3) print(" "\*(n-i),(str(i)+" ")\*(2\*i-1))

Pattern-36:

A

B B B

C C C C C

D D D D D D D E E E E E E E E E

1) n=int(input("Enter the number of rows: "))

2) for i in range(1,n+1):

3) print(" "\*(n-i),(str(chr(64+i)+" "))\*(2\*i-1))

Pattern-37:

A C C C

E E E E E

G G G G G G G I I I I I I I I I



1) n=int(input("Enter the number of rows: "))

2) for i in range(1,n+1):

3) print(" "\*(n-i),(str(chr(64+2\*i-1)+" "))\*(2\*i-1))

Pattern-38:

1

1 2 3

1 2 3 4 5

1 2 3 4 5 6 7

1 2 3 4 5 6 7 8 9

|  |
| --- |
| **1) n=int(input("Enter the number of rows: "))** |
| **2) for i in range(1,n+1):** |
| **3) print(" "\*(n-i),end="")** |
| **4) for j in range(1,2\*i):** |
| **5) print(j,end=" ")** |
| **6) print()** |

Pattern-39:

1

3 2 1

5 4 3 2 1

7 6 5 4 3 2 1

9 8 7 6 5 4 3 2 1

|  |
| --- |
| **1) n=int(input("Enter the number of rows: "))** |
| **2) for i in range(1,n+1):** |
| **3) print(" "\*(n-i),end="")** |
| **4) for j in range(2\*i-1,0,-1):** |
| **5) print(j,end=" ")** |
| **6) print()** |

Pattern-40:

A

A B C

A B C D E

A B C D E F G

A B C D E F G H I

|  |
| --- |
| **1) n=int(input("Enter the number of rows: "))** |
| **2) for i in range(1,n+1):** |
| **3) print(" "\*(n-i),end="")** |
| **4) for j in range(65,65+2\*i-1):** |



|  |
| --- |
| **5) print(chr(j),end=" ")** |
| **6) print()** |

Pattern-41:

A

C B A

E D C B A

G F E D C B A

I H G F E D C B A

|  |
| --- |
| **1) n=int(input("Enter the number of rows: "))** |
| **2) for i in range(1,n+1):** |
| **3) print(" "\*(n-i),end="")** |
| **4) for j in range(65+2\*i-2,64,-1):** |
| **5) print(chr(j),end=" ")** |
| 6) **print()** |

Pattern-42:

0

1 0 1

2 1 0 1 2

3 2 1 0 1 2 3

4 3 2 1 0 1 2 3 4

|  |
| --- |
| **1) n=int(input("Enter the number of rows: "))** |
| **2) for i in range(1,n+1):** |
| **3) print(" "\*(n-i),end="")** |
| **4) for j in range(1,i):** |
| **5) print(i-j,end=" ")** |
| **6) for k in range(0,i):** |
| **7) print(k,end=" ")** |
| 8) **print()** |

Pattern-43:

A

B A B

C B A B C

D C B A B C D

E D C B A B C D E

1) n=int(input("Enter the number of rows: "))

2) for i in range(1,n+1):

3) print(" "\*(n-i),end="")



|  |
| --- |
| **4) for j in range(1,i):** |
| **5) print(chr(i-j+65),end=" ")** |
| **6) for k in range(0,i):** |
| **7) print(chr(k+65),end=" ")** |
| **8) print()** |

Pattern-44:

1

1 2 1

1 2 3 2 1

1 2 3 4 3 2 1

1 2 3 4 5 4 3 2 1

|  |
| --- |
| **1) n=int(input("Enter the number of rows: "))** |
| **2) for i in range(1,n+1):** |
| **3) print(" "\*(n-i),end="")** |
| **4) for j in range(1,i+1):** |
| **5) print(j,end=" ")** |
| **6) for k in range(i-1,0,-1):** |
| **7) print(k,end=" ")** |
| **8) print()** |

Pattern-45:

A

A B A

A B C A B

A B C D A B C

A B C D E A B C D

|  |
| --- |
| **1) n=int(input("Enter the number of rows: "))** |
| **2) for i in range(1,n+1):** |
| **3) print(" "\*(n-i),end="")** |
| **4) for j in range(1,i+1):** |
| **5) print(chr(64+j),end=" ")** |
| **6) for k in range(1,i):** |
| **7) print(chr(64+k),end=" ")** |
| **8) print()** |

Pattern-46:

|  |
| --- |
| **1) n=int(input("Enter a number:"))** |
| **2) for i in range(1,n+1):** |
| **3) print(" "\*(n-i),end="")** |
| **4) for j in range(1,i+1):** |



|  |
| --- |
| **5) print(n+1-j,end=" ")** |
| **6) print()** |

5

5 4

5 4 3

5 4 3 2

5 4 3 2 1

Pattern-47:

|  |
| --- |
| **1) num=int(input("Enter a number:"))** |
| **2) for i in range(1,num+1):** |
| **3) print(" "\*(i-1),end="")** |
| **4) for j in range(1,num+2-i):** |
| **5) print("\*",end=" ")** |
| **6) for k in range(1,num+1-i):** |
| **7) print("\*",end=" ")** |
| **8) print()** |

\* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \*

\* \* \* \* \*

\* \* \*

\*

Pattern-48:

|  |
| --- |
| **1) num=int(input("Enter a number:"))** |
| **2) for i in range(1,num+1):** |
| **3) print(" "\*(i-1),end="")** |
| **4) for j in range(0,num+1-i):** |
| **5) print(num+1-i,end=" ")** |
| **6) for k in range(1,num+1-i):** |
| **7) print(num+1-i,end=" ")** |
| **8) print()** |

5 5 5 5 5 5 5 5 5

4 4 4 4 4 4 4

3 3 3 3 3

2 2 2

1



Pattern-49:

|  |
| --- |
| **1) num=int(input("Enter a number:"))** |
| **2) for i in range(1,num+1):** |
| **3) print(" "\*(i-1),end="")** |
| **4) for j in range(0,num+1-i):** |
| **5) print(2\*num+1-2\*i,end=" ")** |
| **6) for k in range(1,num+1-i):** |
| **7) print(2\*num+1-2\*i,end=" ")** |
| **8) print()** |

9 9 9 9 9 9 9 9 9

7 7 7 7 7 7 7

5 5 5 5 5

3 3 3

1

Pattern-50:

|  |
| --- |
| **1) num=int(input("Enter a number:"))** |
| **2) for i in range(1,num+1):** |
| **3) print(" "\*(i-1),end="")** |
| **4) for j in range(1,num+2-i):** |
| **5) print(j,end=" ")** |
| **6) for k in range(2,num+2-i):** |
| **7) print(num+k-i,end=" ")** |
| 8) **print()** |

1 2 3 4 5 6 7

1 2 3 4 5

1 2 3

1

Pattern-51:

|  |
| --- |
| **1) num=int(input("Enter a number:"))** |
| **2) for i in range(1,num+1):** |
| **3) print(" "\*(i-1),end="")** |
| **4) for j in range(1,num+2-i):** |
| **5) print(chr(65+num-i),end=" ")** |
| **6) for k in range(2,num+2-i):** |
| **7) print(chr(65+num-i),end=" ")** |
| 8) **print()** |



E E E E E E E E E D D D D D D D C C C C C

B B B A

Pattern-52:

|  |
| --- |
| **1) num=int(input("Enter a number:"))** |
| **2) for i in range(1,num+1):** |
| **3) print(" "\*(i-1),end="")** |
| **4) for j in range(1,num+2-i):** |
| **5) print(chr(65+2\*num-2\*i),end=" ")** |
| **6) for k in range(2,num+2-i):** |
| **7) print(chr(65+2\*num-2\*i),end=" ")** |
| 8) **print()** |

I I I I I I I I I

G G G G G G G E E E E E

C C C A

Pattern-53:

|  |
| --- |
| **1) num=int(input("Enter a number:"))** |
| **2) for i in range(1,num+1):** |
| **3) print(" "\*(i-1),end="")** |
| **4) for j in range(1,num+2-i):** |
| **5) print(chr(64+j),end=" ")** |
| **6) for k in range(2,num+2-i):** |
| **7) print(chr(68+k-i),end=" ")** |
| **8) print()** |

A B C D E F G A B C D E

A B C A

Pattern-54:

1) num=int(input("Enter a number:"))

2) for i in range(1,num+1):

3) print(" "\*(num-i),end="")

4) for j in range(1,i+1):

5) print(j,end=" ")



|  |
| --- |
| **6) print()** |
| **7) for k in range(1,num):** |
| **8) print(" "\*k,end="")** |
| **9) for l in range(1,num+1-k):** |
| **10) print(l,end=" ")** |
| **11) print()** |

1

1 2

1 2 3

1 2 3 4

1 2 3 4 5

1 2 3 4

1 2 3

1 2

1

Pattern-55:

**1) num=int(input("Enter a number:"))**

**3) print(" "\*(num-i),end="")**

**5)**

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

**7) for k in range(1,num):**

**9) for l in range(1,num+1-k):**

**11) print()**

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

**10)**

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

**6) print()**

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

**2) for i in range(1,num+1):**

\*

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

\* \* \* \*

\* \* \*

\* \*

\*

Pattern-56:

|  |
| --- |
| **1) num=int(input("Enter a number:"))** |
| **2) for i in range(1,num+1):** |
| **3) print(" "\*(num-i),end="")** |
| **4) for j in range(1,i+1):** |



**5)**

**print(num-j,end=" ")**

**7) for k in range(1,num):**

**9) for l in range(1,num+1-k):**

**11) print()**

**print(num-l,end=" ")**

**10)**

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

**6) print()**

4

4 3

4 3 2

4 3 2 1

4 3 2 1 0

4 3 2 1

4 3 2

4 3

4

Pattern-57:

**1) num=int(input("Enter a number:"))**

**3) print(" "\*(num-i),end="")**

**5)**

**print(num+j-i,end=" ")**

**7) for k in range(1,num):**

**9) for l in range(1,num+1-k):**

**11) print()**

**print(l+k-1,end=" ")**

**10)**

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

**6) print()**

**4) for j in range(0,i):**

**2) for i in range(1,num+1):**

3

2 3

1 2 3

0 1 2 3

1 2 3

2 3

3

Pattern-58:

1) num=int(input("Enter a number:"))

2) for i in range(1,num+1):

3) print(" "\*(num-i),end="")

4) for j in range(0,i):

5) print(chr(65+num+j-i),end=" ")



|  |
| --- |
| **6) print()** |
| **7) for k in range(1,num):** |
| **8) print(" "\*k,end="")** |
| **9) for l in range(0,num-k):** |
| **10) print(chr(65+k+l),end=" ")** |
| **11) print()** |

E D E C D E

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

C D E D E E

Pattern-59:

**1) num=int(input("Enter a number:"))**

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

**5) print()**

**7) for k in range(1,num+1-a):**

9)

**print()**

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

**8)**

**6) for a in range(1,num+1):**

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

**4)**

**2) for i in range(1,num+1):**

\*

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

\* \* \* \*

\* \* \*

\* \*

\*

Pattern-60:

**1) num=int(input("Enter a number:"))**

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

**5) print()**

**7) for k in range(1,num+1-a):**

**6) for a in range(1,num+1):**

**print(num-j,end=" ")**

**4)**

**2) for i in range(1,num+1):**



**9) print()**

**print(num-k,end=" ")**

**8)**

4

4 3

4 3 2

4 3 2 1

4 3 2 1 0

4 3 2 1

4 3 2

4 3

4

Pattern-61:

**1) num=int(input("Enter a number:"))**

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

**5) print()**

**7) for k in range(0,num-a):**

**9)**

**print()**

**print(k+a,end=" ")**

**8)**

**6) for a in range(1,num+1):**

**print(num-i+j-1,end=" ")**

**4)**

**2) for i in range(1,num+1):**

4

3 4

2 3 4

1 2 3 4

0 1 2 3 4

1 2 3 4

2 3 4

3 4

4

Pattern-62:

**1) num=int(input("Enter a number:"))**

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

**5) print()**

**7) for k in range(0,num-a):**

9)

**print()**

**print(chr(65+a),end=" ")**

**8)**

**6) for a in range(1,num+1):**

**print(chr(65+num-i),end=" ")**

**4)**

**2) for i in range(1,num+1):**



E

D D C C C

B B B B

A A A A A B B B B

C C C D D E

Pattern-63:

1) for i in range(1,num+1):

2) for j in range(1,i+1):

3) print(chr(65+num-j),end=" ")

4) print()

5) for a in range(1,num+1):

6) for k in range(num-a,0,-1):

7) print(chr(64+k+a),end=" ")

8) print()

9) num=int(input("Enter a number:"))

E

E D

E D C

E D C B

E D C B A E D C B

E D C E D

E

Pattern-64:

**1) num=int(input("Enter a number:"))**

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

**5) print()**

**7) for k in range(1,num-a+1):**

**9) print()**

**print(chr(64+k+a),end=" ")**

**8)**

**6) for a in range(1,num+1):**

**print(chr(64+num-i+j),end=" ")**

**4)**

**2) for i in range(1,num+1):**



E D E

C D E

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

C D E D E

E

Pattern-65:

|  |
| --- |
| **1) num=int(input("Enter a number:"))** |
| **2) for i in range(1,num+1):** |
| **3) print(" "\*(num-i),end="")** |
| **4) for j in range(1,i+1):** |
| **5) print("\*",end=" ")** |
| 6) **print()** |

\*

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

Pattern-66:

|  |
| --- |
| **1) num=int(input("Enter a number:"))** |
| **2) for i in range(1,num+1):** |
| **3) print(" "\*(num-i),end="")** |
| **4) for j in range(1,i+1):** |
| **5) print(i,end=" ")** |
| 6) **print()** |

1

2 2

3 3 3

4 4 4 4

5 5 5 5 5

Pattern-67:

1) num=int(input("Enter a number:"))

2) for i in range(1,num+1):

3) print(" "\*(num-i),end="")



4) for j in range(1,1+i):

5) print(j,end=" ")

6) **print()**

1

1 2

1 2 3

1 2 3 4

1 2 3 4 5

Pattern-68:

|  |
| --- |
| **1) num=int(input("Enter a number:"))** |
| **2) for i in range(1,num+1):** |
| **3) print(" "\*(num-i),end="")** |
| **4) for j in range(1,1+i):** |
| **5) print(chr(64+i),end=" ")** |
| **6) print()** |

A B B C C C

D D D D E E E E E

Pattern-69:

|  |
| --- |
| **1) num=int(input("Enter a number:"))** |
| **2) for i in range(1,num+1):** |
| **3) print(" "\*(num-i),end="")** |
| **4) for j in range(1,1+i):** |
| **5) print(chr(64+j),end=" ")** |
| **6) print()** |

A A B

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

Pattern-70:

|  |
| --- |
| **1) num=int(input("Enter a number:"))** |
| **2) for i in range(1,num+1):** |
| **3) print(" "\*(i-1),end="")** |
| **4) for j in range(1,num+2-i):** |



|  |
| --- |
| **5) print("\*",end=" ")** |
| 6) **print()** |

\* \* \* \* \*

\* \* \* \*

\* \* \*

\* \*

\*

Pattern-71:

|  |
| --- |
| **1) num=int(input("Enter a number:"))** |
| **2) for i in range(1,num+1):** |
| **3) print(" "\*(i-1),end="")** |
| **4) for j in range(1,num+2-i):** |
| **5) print(num-i+1,end=" ")** |
| 6) **print()** |

5 5 5 5 5

4 4 4 4

3 3 3

2 2

1

Pattern-72:

|  |
| --- |
| **1) num=int(input("Enter a number:"))** |
| **2) for i in range(1,num+1):** |
| **3) print(" "\*(i-1),end="")** |
| **4) for j in range(1,num+2-i):** |
| **5) print(num+2-i-j,end=" ")** |
| 6) **print()** |

5 4 3 2 1

4 3 2 1

3 2 1

2 1

1

Pattern-73:

|  |
| --- |
| **1) num=int(input("Enter a number:"))** |
| **2) for i in range(1,num+1):** |
| **3) print(" "\*(i-1),end="")** |
| **4) for j in range(1,num+2-i):** |
| **5) print(chr(65+num-i),end=" ")** |
| 6) **print()** |



E E E E E D D D D C C C

B B A

Pattern-74:

|  |
| --- |
| **1) num=int(input("Enter a number:"))** |
| **2) for i in range(1,num+1):** |
| **3) print(" "\*(i-1),end="")** |
| **4) for j in range(1,num+2-i):** |
| **5) print(chr(65+num+1-i-j),end=" ")** |
| 6) **print()** |

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

B A A

Pattern-75:

|  |
| --- |
| **1) num=int(input("Enter a number:"))** |
| **2) for i in range(1,num+1):** |
| **3) print(" "\*(i-1),end="")** |
| **4) for j in range(1,num+2-i):** |
| **5) print(chr(64+j),end=" ")** |
| **6) print()** |

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

A B A

Pattern-76:

|  |
| --- |
| **1) num=int(input("Enter a number:"))** |
| **2) for i in range(1,num+1):** |
| **3) print(" "\*(num-i),end="")** |
| **4) for j in range(1,i+1):** |
| **5) print("\*",end=" ")** |
| **6) print()** |
| **7) for p in range(1,num):** |
| **8) print(" "\*p,end="")** |



**9) for q in range(1,num+1-p):**

11) **print()**

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

**10)**

\*

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

\* \* \* \*

\* \* \*

\* \*

\*

Pattern-77:

**1) num=int(input("Enter a number:"))**

**3) print(" "\*(num-i),end="")**

**5)**

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

**7) for p in range(1,num):**

**9) for q in range(1,num+1-p):**

11) **print()**

**print(num-p,end=" ")**

**10)**

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

**6) print()**

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

**2) for i in range(1,num+1):**

1

2 2

3 3 3

4 4 4 4

5 5 5 5 5

4 4 4 4

3 3 3

2 2

1

Pattern-78:

1) num=int(input("Enter a number:"))

2) for i in range(1,num+1):

3) print(" "\*(num-i),end="")

4) for j in range(1,i+1):

5) print(j,end=" ")

6) print()

7) for p in range(1,num):



|  |
| --- |
| **8) print(" "\*p,end="")** |
| **9) for q in range(1,num+1-p):** |
| **10) print(q+p,end=" ")** |
| 11) **print()** |

1

1 2

1 2 3

1 2 3 4

1 2 3 4 5

2 3 4 5

3 4 5

4 5

5

Pattern-79:

**1) num=int(input("Enter a number:"))**

**3) print(" "\*(num-i),end="")**

**5)**

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

**7) for p in range(1,num):**

**9) for q in range(1,num+1-p):**

**11) print()**

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

**10)**

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

**6) print()**

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

**2) for i in range(1,num+1):**

1

1 2

1 2 3

1 2 3 4

1 2 3 4 5

1 2 3 4

1 2 3

1 2

1

Pattern-80:

|  |
| --- |
| **1) num=int(input("Enter a number:"))** |
| **2) for i in range(1,num+1):** |
| **3) print(" "\*(num-i),end="")** |
| **4) for j in range(1,i+1):** |
| **5) print(chr(64+i),end=" ")** |
| **6) print()** |



**7) for p in range(1,num):**

**9) for q in range(1,num+1-p):**

**11) print()**

**print(chr(64+num-p),end=" ")**

**10)**

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

A B B C C C

D D D D E E E E E D D D D

C C C B B A

Pattern-81:

**1) num=int(input("Enter a number:"))**

**3) print(" "\*(num-i),end="")**

**5)**

**print(chr(64+j),end=" ")**

**7) for p in range(1,num):**

**9) for q in range(1,num+1-p):**

11) **print()**

**print(chr(64+q+p),end=" ")**

**10)**

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

**6) print()**

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

**2) for i in range(1,num+1):**

A A B A B C

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

C D E D E E

Pattern-82:

1) n=int(input("Enter a number:"))

2) for i in range(1,n+1):

3) print(" "\*(n-i),end="")

4) for j in range(1,i+1):

5) print(n-i+j,end=" ")



|  |
| --- |
| **6) for k in range(2,i+1):** |
| **7) print(n+1-k,end=" ")** |
| **8) print()** |
| **9) for i in range(1,n+1):** |
| **10) print(" "\*i,end="")** |
| **11) for j in range(1+i,n+1):** |
| **12) print(j,end=" ")** |
| **13) for k in range(2,n+1-i):** |
| **14) print(n+1-k,end=" ")** |
| **15) print()** |

5

4 5 4

3 4 5 4 3

2 3 4 5 4 3 2

1 2 3 4 5 4 3 2 1

2 3 4 5 4 3 2

3 4 5 4 3

4 5 4

5

Pattern-83:

|  |
| --- |
| **1) while True:** |
| **2) n=int(input("Enter a number:"))** |
| **3) for i in range(1,n+1):** |
| **4) print(" "\*(n-i),end="")** |
| **5) for j in range(1,i+1):** |
| **6) print(n+1-j,end=" ")** |
| **7) for k in range(2,i+1):** |
| **8) print(n-i+k,end=" ")** |
| **9) print()** |
| **10) for i in range(1,n+1):** |
| **11) print(" "\*i,end="")** |
| **12) for j in range(1,n+1-i):** |
| **13) print(n+1-j,end=" ")** |
| **14) for k in range(2,n+1-i):** |
| **15) print(i+k,end=" ")** |
| 16) **print()** |

5

5 4 5

5 4 3 4 5

5 4 3 2 3 4 5

5 4 3 2 1 2 3 4 5

5 4 3 2 3 4 5

5 4 3 4 5

5 4 5



5

Pattern-84:

|  |
| --- |
| **1) num=int(input("Enter a number:"))** |
| **2) for i in range(1,num+1):** |
| **3) print(" "\*(num-i),end="")** |
| **4) for j in range(i,i+1):** |
| **5) print("\*",end=" ")** |
| **6) if i>=2:** |
| **7) print(" "\*(2\*i-4),end="")** |
| **8) for k in range(i,i+1):** |
| **9) print("\*",end=" ")** |
| 10) **print()** |

\*

\* \*

\* \*

\* \*

\* \*

Pattern-85:

|  |
| --- |
| **1) num=int(input("Enter a number:"))** |
| **2) for i in range(1,num+1):** |
| **3) print(" "\*(num-i),end="")** |
| **4) for j in range(i,i+1):** |
| **5) print(i,end=" ")** |
| **6) if i>=2:** |
| **7) print(" "\*(2\*i-4),end="")** |
| **8) for k in range(i,i+1):** |
| **9) print(i,end=" ")** |
| **10) print()** |

1

2 2

3 3

4 4

5 5

Pattern-86:

1) num=int(input("Enter a number:"))

2) for i in range(1,num+1):

3) print(" "\*(num-i),end="")

4) for j in range(i,i+1):



|  |
| --- |
| **5) print(num+1-i,end=" ")** |
| **6) if i>=2:** |
| **7) print(" "\*(2\*i-4),end="")** |
| **8) for k in range(i,i+1):** |
| **9) print(num+1-i,end=" ")** |
| 10) **print()** |

5

4 4

3 3

2 2

1 1

Pattern-87:

|  |
| --- |
| **1) num=int(input("Enter a number:"))** |
| **2) for i in range(1,num+1):** |
| **3) print(" "\*(num-i),end="")** |
| **4) for j in range(i,i+1):** |
| **5) print(chr(64+num+1-i),end=" ")** |
| **6) if i>=2:** |
| **7) print(" "\*(2\*i-4),end="")** |
| **8) for k in range(i,i+1):** |
| **9) print(chr(64+num+1-i),end=" ")** |
| 10) **print()** |

E D D C C

B B

A A

Pattern-88:

1. **num=int(input("Enter a number:"))**

|  |
| --- |
| **2) for i in range(1,num+1):** |
| **3) print(" "\*(num-i),end="")** |
| **4) for j in range(i,i+1):** |
| **5) print(chr(64+i),end=" ")** |
| **6) if i>=2:** |
| **7) print(" "\*(2\*i-4),end="")** |
| **8) for k in range(i,i+1):** |
| **9) print(chr(64+i),end=" ")** |
| **10) print()** |



A B B C C

1. **D**
2. **E**

Pattern-89:

|  |
| --- |
| **1) num=int(input("Enter a number:"))** |
| **2) for i in range(1,num+1):** |
| **3) print(" "\*(i-1),end="")** |
| **4) for j in range(i,i+1):** |
| **5) print("\*",end=" ")** |
| **6) if i<=4:** |
| **7) print(" "\*(2\*num-2\*i-2),end="")** |
| **8) for k in range(i,i+1):** |
| **9) print("\*",end=" ")** |
| **10) print()** |

\* \*

\* \*

\* \*

\* \*

\*

Pattern-90:

|  |
| --- |
| **1) num=int(input("Enter a number:"))** |
| **2) for i in range(1,num+1):** |
| **3) print(" "\*(i-1),end="")** |
| **4) for j in range(i,i+1):** |
| **5) print(i,end=" ")** |
| **6) if i<num:** |
| **7) print(" "\*(2\*num-2\*i-2),end="")** |
| **8) for k in range(i,i+1):** |
| **9) print(i,end=" ")** |
| **10) print()** |

1 1

2 2

3 3

4 4

5



Pattern-91:

|  |
| --- |
| **1) num=int(input("Enter a number:"))** |
| **2) for i in range(1,num+1):** |
| **3) print(" "\*(i-1),end="")** |
| **4) for j in range(i,i+1):** |
| **5) print(num-i+1,end=" ")** |
| **6) if i<=4:** |
| **7) print(" "\*(2\*num-2\*i-2),end="")** |
| **8) for k in range(i,i+1):** |
| **9) print(num-i+1,end=" ")** |
| **10) print()** |

5 5

4 4

3 3

2 2

1

Pattern-92:

|  |
| --- |
| **1) num=int(input("Enter a number:"))** |
| **2) for i in range(1,num+1):** |
| **3) print(" "\*(i-1),end="")** |
| **4) for j in range(i,i+1):** |
| **5) print(chr(64+num-i+1),end=" ")** |
| **6) if i<=4:** |
| **7) print(" "\*(2\*num-2\*i-2),end="")** |
| **8) for k in range(i,i+1):** |
| **9) print(chr(64+num-i+1),end=" ")** |
| **10) print()** |

E E

D D

C C B B A

Pattern-93:

|  |
| --- |
| **1) num=int(input("Enter a number:"))** |
| **2) for i in range(1,num+1):** |
| **3) print(" "\*(i-1),end="")** |
| **4) for j in range(i,i+1):** |
| **5) print(chr(64+i),end=" ")** |
| **6) if i<=4:** |



|  |
| --- |
| **7) print(" "\*(2\*num-2\*i-2),end="")** |
| **8) for k in range(i,i+1):** |
| **9) print(chr(64+i),end=" ")** |
| **10) print()** |

1. **A**
2. **B**
3. **C D D E**

Pattern-94:

|  |
| --- |
| **1) num=int(input("Enter a number:"))** |
| **2) for i in range(1,num+1):** |
| **3) print(" "\*(i-1),end="")** |
| **4) for j in range(1,num+1):** |
| **5) print("\*",end=" ")** |
| **6) print()** |

\* \* \* \* \*

\* \* \* \* \*

\* \* \* \* \*

\* \* \* \* \*

\* \* \* \* \*

Pattern-95:

**1) num=int(input("Enter a number:"))**

**3) print(" "\*(num-i),end="")**

**5)**

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

**7) for k in range(1,i+1):**

**9) print()**

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

**8)**

**6) print(" "\*(num-i),end="")**

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

**2) for i in range(1,num+1):**

\* \*

\* \* \* \*

\* \* \* \* \* \*

\* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*



Pattern-96:

|  |
| --- |
| **1) n=int(input("Enter a number:"))** |
| **2) for i in range(1,n+1):** |
| **3) for j in range(1,i+1):** |
| **4) if (i%2!=0 and j%2!=0 )or(i%2==0 and j%2==0):** |
| **5) print("1",end=" ")** |
| **6) else:** |
| **7) print("0",end=" ")** |
| 8) **print()** |

1

0 1

1 0 1

0 1 0 1

1 0 1 0 1

0 1 0 1 0 1

1 0 1 0 1 0 1

Pattern-97:

**1) num=int(input("Enter a number:"))**

**3) print(" "\*(2\*num-i+3),end="")**

**5)**

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

**7) for i in range(1,num+3):**

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

**11) print()**

**13) print(" "\*(2\*num-i),end="")**

**15)**

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

**17) for i in range(1,num+3):**

**19) print("\* "\*3)**

**18) print(" "\*((2\*num)),end="")**

**16) print()**

**14) for j in range(-2,i+1):**

**12) for i in range(1,num+5):**

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

**10)**

**8) print(" "\*(2\*num-i+1),end="")**

**6) print()**

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

**2) for i in range(1,num+1):**



\*

\* \*

\* \* \*

\* \* \* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

\* \* \* \* \* \*

\* \* \* \* \* \* \*

\* \* \* \* \* \* \* \*

\* \* \* \*

\* \* \* \* \*

\* \* \* \* \* \*

\* \* \* \* \* \* \*

\* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \* \*

\* \* \*

\* \* \*

\* \* \*

\* \* \*

\* \* \*

\* \* \*

Pattern-98:

|  |
| --- |
| **1) num=int(input("Enter a number"))** |
| **2) for i in range(1,num+1):** |
| **3) print(" "\*(2\*num-i),end="")** |
| **4) for j in range(1,i+1):** |
| **5) print("\*",end=" ")** |
| **6) print()** |
| **7) for i in range(1,num+1):** |
| **8) print(" "\*(num-i),end="")** |
| **9) for j in range(1,i+1):** |
| **10) print("\*",end=" ")** |
| **11) print(" "\*(num-i),end="")** |
| **12) for k in range(1,i+1):** |
| **13) print("\*",end=" ")** |
| **14) print()** |

\*

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

\* \*



\* \* \* \*

\* \* \* \* \* \*

\* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

Pattern-99:

**1) n=int(input("Enter a number"))**

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

**5) else:**

7)

**print()**

**print("\*"\*(i+1),end=" ")**

**6)**

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

**4)**

**2) for i in range(1,2\*n+1):**

\*\*

\*\*

\*\*\*\*

\*\*\*\*

\*\*\*\*\*\*

\*\*\*\*\*\*

\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*

Pattern-100:

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



\*

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

\* \* \* \* \* \*

\* \* \* \* \* \* \*

\* \* \* \* \* \* \* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

\* \* \* \* \* \*

\* \* \* \* \* \* \*

\* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \*

\* \* \* \* \* \*

\* \* \* \* \* \* \*

\* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \*

\* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \* \*

\* \* \*

\* \* \*

\* \* \*

\* \* \*

\* \* \*

\* \* \*

\* \* \*