INDEX

INTRODUCTION

HISTORY: - Python is created in 20 February 1991. It created by **Guido van rossum**. The name of this language is inspired from comedy series “Monty Python’s Flying Circus”. It is widely used in technical field of machine learning, AI, web dev, mobile & desktop applications, etc.

FEATURES: - python has several features as platform independent, high level language, etc.

High level language: - python is high level language; its codes are easy to understand and code in English.
Interpreted language: - its codes translate to machine codes, statement by statement. That’s why debugging process is easy on python.
Object oriented: - python supports object oriented programming. Making objects of classes is possible in python.
Open source: - source codes of python language are freely available for every user. This language is free to use language.
Indentation: - it use space from left side to define that particular block of code is belong to above statement. There is no extra symbols need to used as curly braces and semicolons.
Easy language: - it has ability to avoid syntax errors. It uses easy English words and fast coding entering.
Portable: - python is platform independent language that, its programs can created on one operating system, and run to another operating systems.
Dynamically typed: - in python, there no need to declare data types of variables. Python interpreter automatically detect data types of variable.
Rich libraries & Frameworks: - python’s have ready to use libraries and frameworks.
Example: - Pandas (data science and machine learning), Django (web framework), tensorflow (AI).
Extensible and embeddable: - C and C++ coding can be used in python, and python’s codes can also use in C and C++.
GUI supports: - designing graphical user interface is possible using python’s PyQt framework.

FIRST PROGRAM: -

- First download and install python idle.
- Then open idle and select new file from file menu.
- Then write this code and save file with .py extension.
  print("Hello World")
- Then execute code on click Run -> Run Module.

START

OUTPUT: - print output in program using “print” function. Write any plain text in double quotes or single quotes for print.

Example: -

```
print("python program")
```

COMMENTS: - these are used for give extra information about code or hide some codes for check output instead of deleting all code. There is a single-line and multi-line comment.

Single-line comment: - it is defines with hash sign (#). Anything after # is considered as comment.

Example: - #this is single-line comment

Multi-line comment: - it is defines with triple single quotes (''') or triple double quotes (""""). Write these quotes at starting and end of comment.

Example:-

```
'''
this is multi-line comment
single quotes
'''
"""
this is multi-line comment
double quotes
"""
```

DATA TYPES: - python has several data types as built-in data type or user-defined data types.

Primary data types: - it is also called built in data types these are available in python.

1) **Numeric: -** it supports numeric type values
    a. **Integer: -** it takes integer values without decimal point. Ex 2, 56, 45.
    b. **Float: -** it takes values with decimal point. Ex 3.4, 5.6, 6.78.
    c. **Complex: -** it takes complex values with real number and Imaginary number with j. Ex 3+4j, 4.5+56j, 6.7+9.8j.
2) **String: -** it also takes string data types. for single or multiple character
3) **List: -** it is group of multiple changeable values.
4) **Tuple: -** it is group of multiple unchangeable values.
5) **Dictionary: -** it is group of multiple values with key and properties.
6) **Bool: -** it takes true or false value.

User-defined data types: - these data types are class and array defined as user requirement.

KEYWORDS: - these are reserved word in python, which already define with special meaning.

False	Await	Else	Import	Pass
None	Break	Except	In	Raise
True	Class	Finally	Is	Return
And	Continue	For	Lambda	Try
As	Def	From	Nonlocal	While
Assert	Del	Global	Not	With
Async	Elif	If	Or	Yield

VARIABLES: - these are containers for store value. It did not need to declare data type of value in python.

Rules for declare variable: -

- Variable must start with alphabet or underscore.
- Numbers can be used anywhere except first character of variable.
- Only special symbol underscore (_) is allowed using in a variable.
- Keywords cannot allow to be used as variables name.
- Variables name are case-sensitive.

Example: -

a=45, b="good"

Print variables: -

```
print(a)
print ("grade: - ",b)
```

format function: - it is function print multiple variables, use curly brackets in string where want to place variables values.

```
a=45
b="good"
print ("grade: - {} is {}".format(a,b))
```

format string: - use f before string value in print function for place variable value in string.

```
a=45
b="good"
print (f"grade: - {a} is {b}")
```

type function: - use type function to see data types of variables.

```
a=45
print(type(a))
```

INPUT: - to take user input python has function called input. It takes input as string. Enter any statement in input argument for show message about input. Store that input in variable.

Example: -

```
a=input("Enter name ")
print ("Name :-",a)
print(type(a))
```

For taking input with other data type it will need to convert input in other data types.

Example: - take input as integer.

```
a=int(input("enter number "))
print("number :-",a)
print(type(a))
```

Example: - take input as float number.

```
a=float(input("enter number "))
print("number :-",a)
print(type(a))
```

Example: - take input as complex number.

```
a=complex(input("enter number "))
print("number :-",a)
print(type(a))
```

OPERATORS

There are operators in python for perform operations on data. Operations are written in expression form to display how operation will perform. Expressions are combination of operand and operators. Operators are signs that will perform operations on operand. Operands are values that get operations done by operators. There are types of operators for perform several types of operations.

A = b + c ------ > expression
A, b, c ------ > operands
=, + ------ > operators

Types of operators:-

1) Arithmetic operators
2) Relational operators
3) Logical operators
4) Bitwise operators
5) Assignment operators
6) Identity operators
7) Membership operators

ARITHMATIC OPERATORS: - these are used for perform common mathematical operations.

Operator	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x – y
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	x % y
**	Exponent	Multiply first number itself time of second number	x**y
//	Floor division	Get division in integer value (truncation division operator)	x//y

Example: -

```
a=23
b=4
c=a+b
print("Addition",c)
c=a-b
print("Subtraction",c)
c=a*b
print("Multiplication",c)
c=a/b
print("Division",c)
c=a%b
```

```
print("Modulus",c)
c=a**b
print("Expoent",c)
c=a//b
print("Floor division",c)
```

RELATIONAL OPERATORS: - these are used for comparison between two values. Result of them is true or false. These operators are also known as comparison operators.

Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Example:-

```
a=5
b=7
c=a<b
print("less than",c)
c=a<=b
print("less than or equals to",c)
c=a>b
print("greater than",c)
c=a>=b
print("greater than or equals to",c)
c=a==b
print("equals to",c)
c=a!=b
print("not equals to",c)
```

LOGICAL OPERATORS: - these operators are used for determine logic between two or multiple conditions.

Operator	Name	Description	Example
&&	Logical and	Returns true if both statements are true	x < 5 && x < 10
||	Logical or	Returns true if one of the statements is true	x < 5 || x < 4
!	Logical not	Reverse the result returns false if the result is true	!(x < 5 && x < 10)

Example:-

```
a=5
b=7
c=a<b and a<5
print("logical and",c)
c=a>b or a==b
print("logical or",c)
c=not a==b
print("logical not",c)
```

BITWISE OPERATORS: - these operators convert decimal values in binary form and perform logical operations on them and return result.

operator	Name	Description	Example
&	Bitwise and	Perform and operation bitwise on values	X&Y
|	Bitwise or	Perform or operation bitwise on values	X|Y
^	Bitwise xor	Perform xor operation bitwise on values	X^Y
~	Bitwise negation	Perform negation operation bitwise on values	~X
<<	Bitwise left shift	Perform left shift operation bitwise on values	X<<Y
>>	Bitwise right shift	Perform right shift operation bitwise on values	X>>Y

Example:-

```
a=5
b=7
c=a&b
print("bitwise and",c)
c=a|b
print("bitwise or",c)
c=a^b
print("bitwise xor",c)
c=~a
print("bitwise negation",c)
c=a<<b
print("bitwise left shift",c)
c=a>>b
print("bitwise right shift",c)
```

ASSIGNMENT OPERATORS: - these operators assign value to variables with arithmetic and bitwise operations.

Operator	Name	Example	Same As
=	Assignment	x = 5	x = 5
+=	add and assign	x += 3	x = x + 3

-=	subtract and assign	x -= 3	x = x – 3
*=	multiply and assign	x *= 3	x = x * 3
/=	divide and assign	x /= 3	x = x / 3
%=	modulus and assign	x %= 3	x = x % 3
//=	Floor division and assign	x//=3	X//=3

Example:-

```
a=5
print("assign a",a)
a+=3
print("Add and assign",a)
a-=3
print("Subtract and assign",a)
a*=3
print("Multiply and assign",a)
a/=3
print("Divide and assign",a)
a%=3
print("Modulus and assign",a)
a//=3
print("Floor divide and assign",a)
```

IDENTITY OPERATORS: - these are used for check values and datatype of variables are same or not.

Operators	**Description**
Is	It get true if values and datatypes of variables are same
Is not	It get true if values and datatypes of variables are not same

Example:-

```
a=4
b="4"
c = a is b
print("is = ",c)
c = a is not b
print("is not = ",c)
```

MEMBERSHIP OPERATORS: - these operators are used for check value of variable is present in other variable or not. These operators are work on collection types only.

Operators	**Description**
In	It get true if values and datatypes of variables are same
Not in	It get true if values and datatypes of variables are not same

Example:-

```
a="in"
b="india"
c = a in b
print("in = ",c)
c = a not in b
print("not in = ",c)
```

CONTROL STATEMENT

DECISION MAKING STATEMENTS: - these statements executes block of code on given condition. It controls flow of program

If statement: - this statement execute block of code if condition is true.

Syntax:-

```
If (condition):
        #block of code
```

Example:-

```
a=23
if(a>10):
   print ("a is greater than 10")
```

If-else statement: - this statement execute "if" block of code when condition is true otherwise execute "else" block of code.

Syntax:-

```
If(condition):
        #block of code
Else:
        #block of code
```

Example:-

```
a=6
if(a>10):
   print ("a is greater than 10")
else:
   print ("a is smaller than 10")
```

Elif statement: - it used when needs to check multiple conditions; it execute block of code when condition is true.

Syntax:-

```
If(condition1):
        #block of code
Elif(condition2):
```

```
        #block of code
…
Elif(conditionN):
        #block of code
Else:
        #block of code
```

Example:-

```
a=67
if(a>90):
   print ("Grade:- A+")
elif(a>80):
   print ("Grade:- A")
elif(a>70):
   print ("Grade:- B+")
elif(a>60):
   print ("Grade:- B")
elif(a>50):
   print ("Grade:- C+")
elif(a>40):
   print ("Grade:- C")
else:
   print("failed")
```

LOOPING STATEMENTS: - are used when need to execute block of code multiple times.

While: - it execute group of statements until condition is true.

Syntax:-

```
Initialization
While(condition):
        #block of code
        Increment/ decrement
```

Example:-

```
a=0
while(a<10):
   print("looping")
   a=a+1
```

Example: - nested while loop

```
i=0
while(i<5):
   j=0
   while(j<5):
      print("1",end=" ")
```

```
    j=j+1
  print()
  i=i+1
```

For loop: - this loop special used for looping through elements collection. It also used with range function.

Syntax: - range (end), start =0 (default), increment=+1 (default), iterate until end-1.

```
 for variable in range (end):
        print(variable)
```

Example:-

```
for i in range(10):
  print(i)
```

Syntax: - range (start, end), increment=+1 (default), iterate until end-1.

```
for variable in range (start,end):
        print(variable)
```

Example:-

```
for i in range(1,10):
  print(i)
```

Syntax: - range (start, end, increment/decrement), iterate until end-1.

```
for variable in range (start, end, increment/decrement):
        print(variable)
```

Example:-

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

Syntax: -

```
for variable in collection:
        print(variable)
```

Example:-

```
a="hello world"
for i in a:
  print(i)
```

Example: - nested for loop

```
for i in range(5):
  for i in range(5):
    print("1",end=" ")
  print()
```

Break statement: - it break loop when executed.

```
i=1
while(i<10):
```

```
    print(i)
    if(i==5):
        break
    i+=1
```

Continue statement: - it continues current iteration when executed.

```
i=1
s=0
while(i<10):
    s=s+1
    i=i+1
    if(i<7):
        continue
    print("i =",i)
    print("sum =",s)
```

FUNCTIONS

Function is the block of code that execute for perform particular task. There are two types of function as built-in and user-defined. Built-in functions are already defined in python library. That used them by calling in program.
Example: - print (), input (), int (), range () etc.

USER-DEFINED FUNCTIONS: - these functions are defined by users to perform particular task. These functions are declares using “def” keyword.

Syntax:-

```
Def function_name(arguments):
        #body of function

Function_name(arguments)   #calling function
```

Example:-

```
def hello():
   print("hello world")

hello()
```

TYPES OF FUNCTIONS: - there are four types of functions in python. Return keyword used for return value from function.

1) Without argument and without return value

Example:-

```
def hello():
   print("function")

hello()
```

2) with argument and without return value

Example:-

```
def add(a,b):
   c=a+b
   print ("addition ",c)

add(4,5)
```

3) Without argument and with return value

Example:-

```
def sub():
    a=7
    b=4
    c=a-b
    return c

z=sub()
print("subtraction ",z)
```

4) with argument and with return value

Example:-

```
def mul(a,b):
    c=a*b
    return c

x=mul(9,5)
print("multiplication ",x)
```

LAMBDA: - A lambda function is a small anonymous function. It can take any number of arguments, but can only have one expression. It is construct that create anonymous function at runtime.

Syntax:-

Variable = lambda arguments: expression

Example:-

```
z=lambda a,b:a+b
print(z(3,4))
```

COLLECTIONS

ARRAY: - it is collection of data elements having same data type. It is single variable which store multiple values of same data type.

Syntax:-

```
import array as module_name

array_name = module_name.array('datatype',[value1, value2,…,valueN])
```

Example:-

```
import array as ar

a=ar.array('i',[18,24,13,42,57])
print(a)
```

Take datatype of array

'i' = integer value
'f' = float value
'd'=double value

For loop in array:-

```
print("using value")
for i in a:
   print(i)

print("using index")
for i in range(len(a)):
   print(a[i])
```

Methods of array:-

Append() :- add new item at end of array.
Syntax: - array_name.append(value)
Example: - a.append(45)

Extend() :- append multiple items in array.
Syntax: - array_name.extend([value1,value2,…,valueN])
Example: - a.extend([42,35,64])

Index() :- return index of given value from array
Syntax: - array_name.index(value)
Example: - c=a.index(6)

Insert() :- insert value at given position or index.
Syntax: - array_name.insert(index,value)
Example: - a.insert(8,345)

Pop() :- remove and return item of given index, remove default if index is not given.
Syntax: - array_name.pop(index)
Example: - c=a.pop(4)

Remove() :- remove given value
Syntax: - array_name.remove(value)
Example: - a.remove(4)

Reverse() :- reverse the order of the items of array.
Syntax: - array_name.reverse()
Example: - a.reverse()

Buffer_info() :- return memory address and length of array.
Syntax: - array_name.buffer_info()
Example: - c=a.buffer_info()

Count() :- return count of given value present in array.
Syntax: - array_name.count(value)
Example: - c=a.count(5)

Len() :- returns length of array.
Syntax: - variable=len(array_name)
Example: - z=len(a)

LIST: - it is ordered, mutable index collection of data elements in which repetition of data is allowed. It is built-in collection datatype in python.

Syntax:-
Variable = [value1,value2,…,valueN]

Example:-
```
list1 = [1, 2, 3, 4, 5]
print (list1)
print(type(list1))

list2 = []   #empty list
```

For loop in list:-

```
print("using values")
for i in list1:
   print(i)

print("using index")
for i in range(len(list1)):
   print(list1[i])
```

Methods of list:-

Len() :- return length of list.
Syntax: - variable = len(list_name)
Example: - l=len(list2)

Insert() :- insert given object at given index.
Syntax: - list_name.insert(index, object)
Example: - list2.insert(8,45)

Append() :- insert object at last position of list.
Syntax: - list_name.append(object)
Example: - list2.append(56)

Extend() :- extent list by appending elements from iterable
Syntax: - list_name.extend(iterable)
Example: - list2.extend(a)

Index() :- return index of given object
Syntax: - list_name.index(object) or list_name.index(object,start,stop) -> find in particular range
Example: - z=list2.index(4)

Count() :- return count of given object.
Syntax: - list_name.count(object)
Example: - z=list2.count(4)

Remove() :- remove given value from list.
Syntax: - list_name.remove(value)
Example: - list2.remove('2')

Pop() :- remove and return object at given index or remove object at last index in default
Syntax: - list_name.pop(index)
Example: - z=list2.pop(4)

Reverse() :- reverse the order of objects in list.
Syntax: - list_name.reverse()
Example: - list2.reverse()

Sort() :- sort objects of list in ascending order, use (reverse=True) for sort in descending order.
Syntax: - list_name.sort() or list_name.sort(reverse=True)

Copy() :- copy list items to left variable.
Syntax: - variable=list_name.copy()
Example: - l=list2.copy()

Clear() :- delete all objects in list.
Syntax: - list_name.clear()
Example: - list2.clear()

TUPLE: - it is ordered, immutable collection of data item in which repetition is allows.

Syntax:-
Var= (value1, value2,…, valueN)

Example:-
```
d = (15, 23, 32, 43, 54)
print(d)
```

For loop in tuple:-

```
for i in d:
   print(i)

for i in range(len(d)):
   print(d[i])
```

Change tuple value:-

```
d=(15,23,32,43,54)
l=list(d)
l.append(34)
d=tuple(l)
print(d)
```

Add new items in tuple:-

```
d = (15, 23, 32, 43, 54)
x=(34,)
d+=x;
print(d)
```

Methods of tuple:-

Index() :- it return index of given value from tuple.
Syntax: - tuple_name.index(value)
Example: - z=d.index(43)

Count() :- return count of given value.
Syntax: - tuple_name.index(value)
Example: - z=d.count(43)

Len() :- return length of tuple.
Syntax: - variable = len(tuple_name)
Example: - l=len(tuple1)

DICTIONARY: - it is mutable, not repeatable collection of data, which contain key and value pair.

Syntax:-
variable = {key1: value1, key2: value2,…, keyN:valueN }

Example:-
```
d1={'name':"ram",'age':21,'marks':76}
print(d1)
```

Access value:-
```
print(d1['age'])    #access value using key
```

For loop in dictionary:-

```
print("display keys")
for i in d1:
   print(i)

print("display values")
for i in d1:
   print(d1[i])

print("display keys using method")
for i in d1.keys():
   print(i)

print("display values using method")
for i in d1.values():
   print(i)

print("display keys and values using method")
for i,j in d1.items():
   print(i,j)
```

Methods:-

Keys(): - return all key from dictionary as objects of list
Syntax: - variable=dict_name.keys()
Example: - z=d1.keys()

Values(): - return all values from dictionary as objects of list
Syntax: - variable=dict_name.values()
Example: - z=d1.values()

Items(): - return keys and values pairs in list as tuple.
Syntax: - variable=dict_name.items()
Example: - z=d1.items()

Get(): - return value of given key from dictionary.
Syntax: - variable=dict_name.get(key)
Example: - z=d1.get('age')

Setdefault(): - add new key and value pair in dictionary.
Syntax: - dict_name.setdefault(key,values)
Example: - d1.setdefault('email','ram21@mail.com')

Pop(): - remove key and value pair of given key.
Syntax: - dict_name.pop(key)
Example: - d1.pop('email')

Popitem(): - remove last key and value pair from dictionary.
Syntax: - dict_name.popitem()
Example: - d1.popitem()

Update(): - it update existing value by key, if key is not present then it add provided key and value pair in dictionary.
Syntax: - dict_name.update(key=value)
Example: - d1.update(name="shyam")

Copy(): - copy dictionary objects to other variable.
Syntax: - variable=dict_name.copy()
Example: - d2=d1.copy()

Clear(): - delete all key and value pairs from dictionary
Syntax: - dict_name.clear()
Example: - d1.clear()

OOP

CLASS AND OBJECTS: - almost everything in python is an object, with properties and methods. Classes are blueprints for creating objects. Create class in python by using keyword “class”. There are methods and properties are contains in class and them also called attributes. Methods are functions of class and properties are variables in class.

Syntax:-

```
Class class_name:
        #properties and methods
Object_name = class_name()
```

Example:-

```
class digits:
   x=5
   y=5

o= digits()
print(o.x)
```

Methods: - there are functions define in class, which only callable through object of class.

Syntax:-

```
Def method_name(paramters):
        #code in method
```

Example:-

```
class digits:
   x=None
   y=None
   def show(s):
      print("x=",s.x)
      print("y=",s.y)


o= digits()
o.x=5
o.y=10
o.show()
```

Self: - The self parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class.

None: - it is key word for declare properties without assigning values.

CONSTRUCTOR: - it invoke when object is created. To define constructors use built in function "__init__()" . Parameters in this function take values when class of object is created.

Syntax:-

```
Class class_name:
        Def __init__(parameters):
                #codes in constructor
```

Example:-

```
class car:
   name=None
   model=None
   def __init__(self,n,m):
      self.name=n
      self.model=m

   def display(self):
      print("car name=",self.name)
      print("car model=",self.model)

c1=car("honda","800")
c1.display()
```

INHERITANCE: - Inheritance allows us to define a class that inherits all the methods and properties from another class.
Parent class is the class being inherited from, also called base class.
Child class is the class that inherits from another class, also called derived class.

Syntax: -

```
Class parent_class:
        #parent class code

Class child_class(parent_class):        #inherits parent class
        #child class code
```

Example: -

```
class vehicle:  #parent class
   name=None
   model=None
   def __init__(self,n,m):
```

```
        self.name=n
        self.model=m

class car(vehicle):
    def display(self):
        print("car name=",self.name)
        print("car model=",self.model)


c1=car("maruti","800")
c1.display()
```

Constructors in inheritance: - creating constructors in both classes will override parent class constructor by child class constructor. For avoiding it need to call parent constructor in class.

Syntax: -

```
Class parent_class:
        #parent class code

Class child_class(parent_class):        #inherits parent class
        Def __init__(parameters):
                Parent_class.__init__(parameters)
        #child class code
```

Example:-

```
class person:  #parent class
    name=None
    age=None
    def __init__(self,n,a):
        self.name=n
        self.age=a

class student(person):
    def __init__(self,n,a,r,p):
        person.__init__(self,n,a)
        self.roll=r
        self.percent=p

    def result(self):
        print("Name = ",self.name)
        print("Age = ",self.age)
        print("student roll no=",self.roll)
        print("car model=",self.percent)

s1=student("ram",23,1,55.67)
s1.result()
```

Super () function: - it will also call parent class constructor.

Example:-

```
class person:  #parent class
   name=None
   age=None
   def __init__(self,n,a):
      self.name=n
      self.age=a

class student(person):
   def __init__(self,n,a,r,p):
      super().__init__(n,a)                #using super function
      self.roll=r
      self.percent=p

   def result(self):
      print("Name = ",self.name)
      print("Age = ",self.age)
      print("student roll no=",self.roll)
      print("car model=",self.percent)


s1=student("ram",23,1,55.67)
s1.result()
```

Pass keyword: - it is used for define blank class or function

Example:-

```
class person():
   name='unknown'
   def good(self):
      pass

   def show(self):
      print("Name=",self.name)

class student(person):
   pass


o=student()
o.name="xyz"
o.show()
```

Types of inheritance:-

Single inheritance:-
Syntax:-

```
Class parent:
        #Parent class code

Class child (parent):
        #child class code
```

Multilevel inheritance:-
Syntax:-

```
Class grandparent:
        #grandparent class code

Class parent (grandparent):
        #Parent class code

Class child (parent):
        #child class code
```

Multiple inheritance:-
Syntax:-

```
Class parent1:
        #parent1 class code
Class parent2:
        #parent2 class code
Class child (parent1, parent2):
        #child class code
```

FILE HANDLING

Open: - it is function for opening file with different modes, it takes two argument first is opening file name and second is mode of file.

Syntax:-
Variable=open(“file_name”,”mode”)

MODES: - these are used for opening file for different operations. It used with open function.

Mode	Name	Description
"r"	Read	Default value. Opens a file for reading, error if the file does not exist
"a"	Append	Opens a file for appending, creates the file if it does not exist
"w"	Write	Opens a file for writing, creates the file if it does not exist
"x"	Create	Creates the specified file, returns an error if the file exists

WRITE: - it is used for write file with write or append mode. It used write function for write in file. Close file with close function when operations on file completed.

Example:-
```
f=open("0test.text","w")
f.write("hello here write\n goood day")
f.close()
print("file write")
```

READ: - it is used for read file with read mode.

Example:-
```
f=open("0test.text","r")
print(f.read())
f.close()
print("\nfile read")
```

DELETE: - delete file with importing OS module in python, with remove method.

```
import os
if os.path.exists("0text.text"):
  os.remove("0text.text")
  print("file deleted")
```

```
else:
  print("The file does not exist")
```

MODULE

A module is a file containing Python definitions and statements. Modules in python are built-in code files. There are some built-in modules in python for perform tasks. Programmers can also create their own user-defined modules and used code from module whenever it need in any other program.

CREAT MODULE: - create module using .py extension in file. And import it in another file.

Example: - arithmetic.py

```
def add(a,b):
   return a+b

def sub(a,b):
   return a-b

def mult(a,b):
   return a*b

def div(a,b):
   return a/b
```

IMPORT MODULE: - import module in any python program using “import” keyword.

Syntax:-

1) **import module_name :-** import module in program and use modules function with module name and function name.
   Example: - module_name.function_name

2) **import module_name as new_name: -** import module in program and give new name to module for use it, use modules function with module new name and function name.
   Example: - new_name.function_name

3) **from module import *: -** import module and use modules functions without module_name or new_name
   Example: - function_name

Example: - main.py

```
import arithmetic as arith
```

```
a=7
b=4.0
c=arith.add(a,b)
print("addition of {} and {} is {}".format(a,b,c))

c=arith.sub(a,b)
print("subtraction of {} and {} is {}".format(a,b,c))

c=arith.mult(a,b)
print("multiplication of {} and {} is {}".format(a,b,c))

c=arith.div(a,b)
print("division of {} and {} is {}".format(a,b,c))
```

EXCEPTION HANDLING

The **try** block lets you test a block of code for errors.
The **except** block lets you handle the error.
The **else** block lets you execute code when there is no error.
The **finally** block lets you execute code, regardless of the result of the try- and except blocks.

TRY AND EXCEPT:-

Example:-

```
try:
   a
   b=45
   print(a+b)

except:
   print('error')
```

example2:-

```
try:
   a=34
   b
   print(a+b)
except Exception as e:          //using exception class for print exception
   print(e)
```

Multiple exceptions: - use name of particular exception that will need to handle

```
except NameError:
   print("not defined")
except:
   print('error')
```

else keyword:- if exception is not occurs

```
try:
   a=3
   b=45
   print(a+b)

except:
```

```
    print('error')
else:
    print("no error")
```

FINALLY: - it executes code from function after function returns values.

```
def division(a,b):
    try:
        print("division of {} and {} is {}".format(a,b,a/b))
        return True
    except:
        print("Error ocured")
        return False
    finally:
        print("function executed")


x=division(34,'s')
print("result:-",x)
```

RAISE AN EXCEPTION: - it is used for make user defined exceptions.

```
a=3
b=0
if(b==0):
    raise Exception("divided by 0")
print(a/b)
```

Type error:-

```
x = "hello"

if not type(x) is int:
  raise TypeError("Only integers are allowed")
```