**PYTHON**

Why U learn python?

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

1. Less code

2. simple and easy to Learn

3. Freeware and open source

4. Dymaically typed programming Language

5. platform independent

6. python is a interpreter

7. python supports Large No. of libraries

* write a program to print september 2022 calendar

import calendar

print(calendar.month(1947,8))

* write a program to print calendar for year of 2022

import calendar

print(calendar.calendar(2022))

* write a program to find the addition of two numbers

using C Language:

#include<stdio.h>

main()

{

int a,b,c; ==>variable declaration statement

a=10;

b=5;

c=a+b;

printf("%d",c);

}

using c++:

#include<iostream.h>

main()

{

int a,b,c; ==> variable declaration statement

a=10;

b=5;

c=a+b;

cout<<c;

}

java:

import java.util.scanners;

class addition

{

public static void main(string args[]

{

int a,b,c; ==>variable declaration statement

a=10;

b=5;

c=a+b;

system.out.println(c);

}

}

In java, Total No. of statements ===> 12 lines

python:

method-1:

a=10

b=5

c=a+b

print(c)

method-2:

a=10

b=5

print(a+b)

method-3:

a,b = 10,15

print(a+b)

method-4:

print( (a:=10) + (b:=15) )

Introduction:

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

1. python is a high level programming Langauge.

2. python is a oop(object-oriented programming) Language

3. python is a pop(procedural oriented programming) Language

4. python is a scripting Language

5. python is a dynamic Language

i.e. No need of varaible declaration

6. python is a interpreter based programming Language

programming Languages are divided into 2 types:

1. static typed

2. dynamic typed

1. static typed:

==> The variable declaration is compulsory.

It is called static typed programming Langauges.

ex: c, c++,java,cobol,pascal,.....etc.

2. dynamically typed:

==> The variable declaration is not required.

It is called dynamically typed programming Langauges.

ex: python,javascript,R, ....etc.

programming Languages are divided into 2 types:

1. compiler

2. interpreter

1. compiler:

It is a langauge translator.

It is used to convert,

high level program into low level program.

To conver the entire program into only once.

2. interpreter

It is a langauge translator.

It is used to convert,

high level program into low level program.

But the conversion is line by line.

variable:

==> It is a value.

==> The value can be changed during execution of the program.

It is called variable.

ex: date,time,age,weather,weight, ....etc.

constant:

==> It is a value.

==> The value cannot be changed during execution of the program.

It is called constant.

ex: surname,name,color,pi ,e .....etc.

Data types

int --> It allows only numbers.

float --> It allows numbers with decimal point

str --> It allows numbers/alphabets/symbols

complex --> a+bj format.

bool --> True/False.

**Types of variables**

**or**

**Data Types**

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

student data:

htno ---> 109ej181901 --> str

name --> venkat --> str

m1 --> 77 --> int

m2 --> 88 --> int

m3 --> 99 --> int

total --> 270 --> int

avg --> 88.00 --> float

employee data:

empid --> 58945787 --> only numbers --> int

ename --> siva --> only alphabets --> str

sal --> 96000.50 --> numbers with decimal point

float

company name --> tronix technologies --> str

department --> admin --> str

bank data:

accno --> 48557584548 --> only numbers --> int

cname --> ram kumar --> only alphabets --> str

balance --> 6500.70 --> numbers with decimal point

float

address -->

H.no: MIG-85/A/96

street: near jntu signal --> str

dist: hyderbasd

state: telangana

pin :500085.

phone --> 9949999999 --> only number --> int

==> python supports the following built-in data types:

1. int

2. float

3. str

4. complex

5. bool

6. list

7. tuple

8. set

9. dict

10. frozenset

11. bytes

12. bytearray

13. None

14. range

1. int data type:

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

==> The int represents integer data type.

==> It allows only digits(numbers).

ex-1:

a = 25

print(a)

print(type(a))

ex-2:

b = 100

print(b)

print(type(b))

2. float data type:

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

==> It allows digits with decimal point or e notation.

ex-1: using decimal point

a = 25.00

print(a)

print(type(a))

ex-2: using decimal point

b = 7.5

print(b)

print(type(b))

ex-3: using e notation

m = 12e4

print(m)

print(type(m))

ex-4:

x = 63e2

print(x)

print(type(x))

**strings**

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

==> It allows numbers/alphabets/symbols.

==> A single character or group of characters,

is called a string.

==> In python , the strings are enclosed by either

single quotes or double quotes or

triple single quotes or triple double quotes.

ex:

'Tronix'

"Tronix"

'''Tronix'''

"""Tronix"""

==> python supports 2 types of strings.

1. single line string

2. multi-line string

1. single line string:

==> A string written in only one line.

It is called single line string.

==> single line strings are enclosed by either

single quotes or double quotes or

triple single quotes or triple double quotes.

ex-1:

'python is a high level programming Language'

ex-2:

"python is a scripting Langauge"

ex-3:

'''python is a oop Language'''

ex-4:

"""python is a dynamic Langauge"""

ex-5: using single quotes

s1 = 'Hello'

print(s1)

print(type(s1))

ex-6: using double quotes

s1 = "Hello"

print(s1)

print(type(s1))

ex-7: using triple single quotes

s1 = '''Hello'''

print(s1)

print(type(s1))

ex-8: using triple double quotes

s1 = """Hello""""

print(s1)

print(type(s1))

2. multi-line string:

==> A string written in morethan one line.

It is called multi-line string.

==> multi-line strings are enclosed by either

triple single quotes or triple double quotes.

ex-1:

'''python is

a high level

programming

Language'''

ex-2:

"""python

is a

scripting Langauge"""

ex-3: using single quotes

s1 = 'Hello

Good

Morning

To

All'

print(s1)

print(type(s1))

Note: It returns syntax error.

i.e. EOL --> End Of Line

ex-4: using double quotes

s1 = "Hello

Good

Morning

To

All"

print(s1)

print(type(s1))

Note: It returns syntax error.

i.e. EOL --> End Of Line

ex-5:

s1 = '''Hello

Good

Morning

To

All'''

print(s1)

print(type(s1))

ex-6:

s1 = """Hello

Good

Morning

To

All"""

print(s1)

print(type(s1))

**arithmetic operators in strings:**

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

1. string concatenation operator(+)

2. string repetition operator(\*)

1. string concatenation operator(+) :

==> This operator is used to concatenate(joining/merging)

of 2 or more strings.

ex-1:

s1 = 'python'

s2 = 'tronix'

s3 = 'hyderabad'

s4 = s1+s2+s3

print(s4)

ex-2:

s1 = 'python'

s2 = 'tronix'

s3 = 'hyderabad'

s4 = s1+' '+s2+' '+s3

print(s4)

ex-3:

s1 = 'python'

s2 = 3.10

s3 = s1+s2

print(s3)

TypeError: can only concatenate str (not "float") to str

ex-4:

s1 = 'python'

s2 = 3

s3 = s1+s2

print(s3)

TypeError: can only concatenate str (not "int") to str

ex-5:

s1 = 'python'

s2 = '3.10'

s3 = s1+s2

print(s3)

2. string repetition operator(\*)

==> This oeprator is used to repeat a string

'n' number of times.

ex-1:

s1 = 'Tronix'

print(s1\*5)

ex-2:

s1 = 'python'

print(s1\*100)

what is Environment? Different types of environments:

==> It is a place(area)

==> Here, we can develop the python code.

==> python supports 3 types of environments:

1. IDLE

2. IDE

3. online environment(online editor)

1. IDLE:

==> IDLE stands for Integrated Development Learning Environment.

==> Here, we can develop the only python programs.

2. IDE:

==> IDE stands for Integrated Development Environment.

==> Here, we can develop python programs,

including HTML,CSS,JAVASCRIPT,C,c++,java,

Django,Flask,react js,angular js,nodejs,

SQL,ORACEL,NOSQL,MYSQL,TSQL,

jquery,bootstrap,.....etc.

==> The Top 10 IDE's are:

pycharm

|

|--> pycharm community edition

|--> pycharm professional edition

Anaconda

|

|--> jupyter lab

|--> jupyter notebook

|--> spyder(scientific python

development environment)

visual studio code(vscode)

Atom

vim

sublime text

webstrom

notepad

notepad++

Thonny

How to download IDLE:

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

www.python.org

|

|

downloads

|

|

windows

|

|

python version 3.10.7

3. online environment(online editor)

==> No need of software downloading and no need of installation.

==> we can use directly in the google.

**operators in python**

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

what is operator?

==> It is a mathematical symbol.

==> It is used eihter calculation/comparison purpose.

ex -1: a+b

a,b are called variables/operands/objects.

+ is called operator.

ex-2: x<y

x,y are called variables/operands/objects

< is called operator.

==> python supports 6 types of operators:

1. arithmetic operators

2. relational operators

3. Logical operators

4. bitwise operators

5. assignment operators

6. special operators

1. arithmetic operators:

==> These operators are used for 'calculation' purpose.

==> It returns value.

operator meaning

+ addition operator

- minus operator

\* mutlplication operator

/ division operator

% remainder or modulo operator

// floor division operator

\*\* power operator or exponent operator

ex-1:

a=5

b=3

print(a+b)

print(a-b)

print(a\*b)

print(a/b)

print(a%b)

print(a//b)

print(a\*\*b)

ex-2:

x=24

y=5

print(x+y)

print(x-y)

print(x\*y)

print(x/y)

print(x%y)

print(x//y)

print(x\*\*y)

2. relational operators:

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

==> These operators are used for 'comparison' purpose.

==> It returns either True/False.

==> These operators are also comparison operators.

operator meaning

< less than operator

> greater than operator

<= lessthan or equal to operator

>= greater than or equal to operator

== equal to operator

!= not equal to operator

ex-1 :

a=10

b=3

print(a<b)

print(a>b)

print(a<=b)

print(a>=b)

print(a==b)

print(a!=b)

ex-2:

x=24

y=5

print(x<y)

print(x>y)

print(x<=y)

print(x>=y)

print(x==y)

print(x!=y)

**arithmetic expression**

==> A combination of arithmetic operators and operands

is called a arithmetic expression.

==> It returns value.

a+b,a-b,a\*b,b\*b-4\*a\*c

relational expression:

==> A combination of relational operators and operands

is called a relational expression.

==> It is also called boolean expression or condition.

==> It returns True/False.

ex: a<b,a>b,a==b

3. Logical operators:

==> These operators are used for,

to connect 2 or more relational expressions.

==> The logical operators are 3:

operator meaning

and logical and operator

or logical or operator

not logical not operator

logical and operator(and):

==> if both relational expressions are True,

then, result is True.

otherise result is False.

==> Truth table:

RE-1 RE-2 RESULT

T T T

T F F

F T F

F F F

ex:

>>> True and True

True

>>> True and False

False

>>> False and True

False

>>> False and False

False

>>>

logical or operator(or):

==> if any one relational expression returns True,

then, result is True.

otherise result is False.

==> Truth table:

RE-1 RE-2 RESULT

T T T

T F T

F T T

F F F

ex-1 :

>>> True or True

True

>>> True or False

True

>>> False or True

True

>>> False or False

False

>>>

logical not operator:

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

==> If relational expression returns True, then result is False.

==> If relational expression returns False, then result is True.

==> truth table:

RE RESULT

T F

F T

ex-1:

>>> not True

False

>>> not False

True

ex: write a program to implement logical operators

a = 10

b = 2

c = 25

print(a<b and a==c)

print(c>a and b<c)

print(c==a and b<c)

print(c==a or b<c)

print(not a<b)

4. bitwise operators:

==> These opeartors are used to perform operations at

binary level.

==> bit stands for binary digit.

==> It represents either 0 or 1.

==> Memory measurents:

1 byte = 8 bits

1 KB = 1024 bytes

1 MB = 1024 KB

1 GB = 1024 MB

1 TB = 1024 GB,........etc.

Assignment operators:

==> These operators are used to store the value into a variable.

ex: x = 99

operator meaning example

= simple assignment operator x = 5

+= addition assignment operator x=x+10 or x+=10

-= minus assignment operator x=x-10 or x-=10

= multiplication assignment operator x=x\*25 or x=25

/= division assignment operator x=x/35 or x/=35

%= modulo assignment operator x=x%35 or x%=35

//= floor division assignment operator x=x//35 or x//=35

\*= power assignment operator x=x5 or x\*=5

ex-1:

m = 9

print(m)

m+=3

print(m)

m-=10

print(m)

m\*\*=5

print(m)

m%=5

print(m)

m\*=6

print(m)

m//=3

print(m)

m/=3

print(m)

type():

It returns data type of the given variable.

ex-1:

m = 25

print(m)

print(type(m))

ex-2: '

x = 4.8

print(x)

print(type(x))

ex-3:

name = 'tronix python 3.10.5'

print(name)

print(type(name))

id():

It returns address of the given variable.

ex-1:

m = 25

print(m)

print(type(m))

print(id(m))

ex-2:

x = 4.8

print(x)

print(type(x))

print(id(x))

ex-3:

name = 'tronix python 3.10.5'

print(name)

print(type(name))

print(id(name)) '

In java:

a = 10

b = 25

c = 10

d = 10

e = 25

f = 10

g = 45

In java, memory requirement: 7 x 4 bytes = 28 bytes

In python:

a = 10

b = 25

c = 10

d = 10

e = 25

f = 10

g = 45

In python, memory requirement: 3 x 4 bytes = 12 bytes

1. write a program to display addresses of the given variables

a = 10

b = 25

c = 10

d = 10

e = 25

f = 10

g = 45

print(id(a))

print(id(b))

print(id(c))

print(id(d))

print(id(e))

print(id(f))

print(id(g))

2. write a program to display addresses of the given variables

x = 5

y = 5

z = 5

print(id(x))

print(id(y))

print(id(z))

special operators:

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

==> python supports 2 types of special operators:

1. identity operators

2. membership operators

1. identity operators:

==> These operators are used to find the both objects

are pointing same location or different location.

==> identity operators are 2:

is --> It returns True/False.

if both objects are pointing same location, then

it returns True. otherwise False.

is not --> It returns True/False.

if both objects are pointing different location, then

it returns True. otherwise False.

ex-1:

x = 5

y = 5

z = 5

a = 45

print(a is x)

print(x is y)

print(z is not y)

ex-2:

s1= 'tronix'

s2 ='tronix'

print(id(s1))

print(id(s2))

print(s1 is s2)

2. Membership operators:

==> These operators are used,

to find the given value is present or not .

==> The membership oeprators are 2 types:

in --> It returns either True/False.

if the given value is present, then

it returns True. otherwise it returns False.

not in --> It returns either True/False.

if the given value is not present, then

it returns True. otherwise it returns False.

ex-1:

s = 'tronix'

print('r' in s)

print('z' in s)

print('N' in s)

print('i' not in s)

ex-2:

s = 'python is a high level programming Language'

print('m' in s)

print('n' in s)

print('java' in s)

print('angu' in s)

ex-3:

x = [11,22,33,44,55,66]

print(22 in x)

print(55 not in x)

print(99 in x)

Methods of assigment:

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

To store the value into a variable.

It is called assignment.

method-1: single variable , single value

a = 25

print(a)

method-2: single variable , multiple values

a = 25,3,99,11,33

print(a)

print(a[0])

print(a[1])

print(a[2])

print(a[3])

print(a[4])

print(a[5])

method-3: multiple varaibles, same value

ex-1:

a=b=c=d=e=11

print(a)

print(b)

print(c)

print(d)

print(e)

ex-2:

i=j=k=sum=count=0

print(i)

print(j)

print(k)

print(sum)

print(count)

method-4: multiple varaibles, multiple values

ex-1:

a,b,c,d,e = 11,22,33,44,55

print(a)

print(b)

print(c)

print(d)

print(e)

Note:

The No. of variables and the No. of values

must be equal .

otherwise it returns Error.

write a program swapping of two numbers

swapping means interchanging of two values.

a = 10

b = 20

after swapping

a=20

b=10

method-1: with temporary variable

a=10

b=20

print('Before swapping')

print('A value = ',a)

print('B value = ',b)

temp = a

a = b

b = temp

print('After swapping')

print('A value = ',a)

print('B value = ',b)

method-2: with temporary variable

a=10

b=20

print('Before swapping')

print('A value = ',a)

print('B value = ',b)

a=a+b

b=a-b

a=a-b

print('After swapping')

print('A value = ',a)

print('B value = ',b)

method-3: to write logic in only one line(python style)

a=10

b=20

print('Before swapping')

print('A value = ',a)

print('B value = ',b)

a,b = b,a

print('After swapping')

print('A value = ',a)

print('B value = ',b)

Types of Numbers:

==> python suppports 4 types of numbers:

1. decimal numbers

2. binary numbers

3. octal numbers

4. Hexa-decimal numbers

1. decimal number system :

It allows 0 ,1,2,3...9.

The Total No. of digits are 10. so its base value '10'.

By default, every number is decimal number.

2. binary number system :

It allows only 2 digits. ie. 0 ,1.

The Total No. of digits are 2. so its base value '2'.

binary number start with '0b' or '0B'.

3. octal number system :

It allows 0 ,1,2,3...7.

The Total No. of digits are 8. so its base value '8'.

octal number start with '0o' or '0O'.

4. Hexa-decimal number system :

==> It allows 0,1,2,3...9.

and lower case letters are : a,b,c,d,e,f.

Upper case letters are : A,B,C,D,E,F.

The value of 'A' or 'a' is 10.

The value of 'B' or 'b' is 11.

The value of 'C' or 'c' is 12.

The value of 'D' or 'd' is 13.

The value of 'E' or 'e' is 14.

The value of 'F' or 'f' is 15.

The Total No. of digits are 16. so its base value '16'.

Hexa-decimal number start with '0x' or '0X'.

ex-1:

To convert:

decimal --> binary decimal --> octal decimal --> Hexa-deciaml

28 --> 0b11100 28 --> 0o34 28 --> 0x1c

ex-2:

To convert:

decimal --> binary decimal --> octal decimal --> Hexa-deciaml

60 --> 0B111100 60 --> 0o74 60 --> 0x3c

>>> bin(60)

'0b111100'

>>> oct(60)

'0o74'

>>> hex(60)

'0x3c'

ex:

10110111 ---> decimal

0b10110111 ---> binary

0o10110111 --> octal

0x10110111 --> Hexa-decimal

ex: 569 --> decimal number

0b1569 --> Error, because '5' is not allowed.

0o569 --> Error, because '9' is not allowed.

0x569 --> hexa-decimal number

Type casting

or

Type conversion

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

==> To convert the values from one data type into another data type

This process is called type casting.

==> The following built-in functions are used for type casting

purpose:

1. int()

2. float()

3. str()

4. complex()

5. bool()

6. bin()

7. oct()

8. hex()

1. int():

==> This function is used to convert the other values

into integer data type, except complex numbers.

ex:

>>> int(9.3)

9

>>> int(125.584583478)

125

>>> int(True)

1

>>> int(False)

0

>>> int('56')

56

>>> int('125')

125

>>> int('python')

ValueError: the value must be decimal number

>>> int(9+4j)

TypeError: can't convert complex to int

2. float():

==> This function is used to convert the other values

into float data type, except complex numbers.

ex:

>>> float(9)

9.0

>>> float(125)

125.0

>>> float(True)

1.0

>>> float(False)

0.0

>>> float('34')

34.0

>>> float('python')

ValueError: could not convert string to float: 'python'

>>> float(7+6j)

TypeError: can't convert complex to float

3. str():

This function is used to convert other values

into string data type.

ex:

>>> str(9)

'9'

>>> str(125.65)

'125.65'

>>> str(True)

'True'

>>> str(False)

'False'

>>> str(5+9j)

'(5+9j)'

>>> str(5-9j)

'(5-9j)'

6. bin():

==> This function is used to convert,

the decimal number into binary number.

7. oct():

==> This function is used to convert,

the decimal number into octal number.

8. hex():

==> This function is used to convert,

the decimal number into hexa-decimal number.

ex-1:

>>> bin(125)

'0b1111101'

>>> oct(125)

'0o175'

>>> hex(125)

'0x7d'

ex-2:

>>> bin(89)

'0b1011001'

>>> oct(89)

'0o131'

>>> hex(89)

'0x59'

input and output statements in python

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

input ---> process(calculation) --> output cycle

AIM input process output

1. To prepare sugar

tea water heat tea

milk

tea powder

2. To prepare grapes

grapes sugar grinding juice

juices milk

ice

3. addition of a = 10

two nubmers b = 20 c = a+b print(c)

4. area of a len = 8

rectangle bre = 5 area = len\*bre print(area)

print():

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

==> It is a built-in function.

==> This function is used to,

display the messages or values on the screen.

==> syntax:

print(value,sep,end,file,flush)

here,

==> value indicates the messages or variable values.

==> sep indicates separtor

==> end indicates end of the statement

==> file, flush are used for file handling concepts.

ex-1: To print the messages

print('Good Morning To All')

ex-2: To print the message

print('Good Morning To All')

print('welcome to tronix technologies')

print('welcome to python course')

ex-3: To print the values

a=10

b=20

c=30

print(a)

print(b)

print(c)

print(a,b,c)

ex-4: To print the values

empid = 1901

ename = 'ram'

sal = 96000

print(empid)

print(ename)

print(sal)

ex-5: To print the values with messages

empid = 1901

ename = 'ram'

sal = 96000

print('Employee Number = ',empid)

print('Employee Name = ',ename)

print('Employee salary = ',sal)

ex-6: To print the values with messages

htno = 45385348

sname = 'prasad'

m1=77

m2=88

m3=99

print('Student Hall ticket Number = ',htno)

print('Student Name = ',sname)

print('First subject Marks = ',m1)

print('Second subject Marks = ',m2)

print('Third subject Marks = ',m3)

print('Total Marks = ',m1+m2+m3)

print('Average Marks = ',(m1+m2+m3)/3)

ex-7: using sep parameter i.e. separator to print date

day = 13

month=10

year=2022

print(day,month,year)

print(day,month,year,sep='-')

print(day,month,year,sep='/')

ex-8: using sep parameter i.e. separator to print time

h = 11

m = 35

s = 22

print(h,m,s)

print(h,m,s, sep='-')

print(h,m,s,sep='/')

print(h,m,s, sep=':')

ex-9: using sep parameter to print strings

s1 = 'python'

s2 = 'tronix'

s3 = 'hyderabad'

print(s1,s2,s3)

print(s1,s2,s3,sep=' ')

print(s1,s2,s3,sep='/')

print(s1,s2,s3,sep='-')

print(s1,s2,s3,sep='&')

print(s1,s2,s3,sep='^')

print(s1,s2,s3,sep=':')

ex-10: using end parameter

s1 = 'python'

s2 = 'tronix'

print(s1,end='')

print(s2)

ex-11: using end parameter '

s1 = 'python'

s2 = 'tronix'

print(s1,end=' ')

print(s2)

ex-12: using end parameter including tab spaces

s1 = 'python'

s2 = 'tronix'

print(s1,end='\t\t\t\t')

print(s2)

# Note: \t ---> tab spaces i.e. 4 blank spaces

ex-13: using end parameter with new line

s1 = 'python'

s2 = 'tronix'

print(s1,end='\n\n\n\n\n\n' )

print(s2)

# Note: \n ---> new line or next line .

ex-14:

s1 = 'python'

s2 = 'tronix'

print(s1)

print()

print()

print()

print()

print()

print()

print(s2)

ex-15: To display 4 empty lines/ 4 new lines

print()

print()

print()

print()

input():

==> It is a built-in function.

==> It is used to get the used input.

==> syntax:

variablename = input('Any prompt ...')

ex-1:

x = input('Enter your Name')

ex-2:

age = input('Enter your age')

**programs**

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

1. write a program to print Hello World

print('Hello World !!!!!! ')

2. write a program to find the addition of two numbers

a = int(input('Enter first number'))

b = int(input('Enter second number'))

c = a + b

print(c)

print('Addition of two numbers = ',c)

3. write a program to find the area of a rectangle

Hint : area = length x breadth

l = int(input('Enter length value'))

b = int(input('Enter breadth value'))

area = l \* b

print('Area of a rectangle = ',area)

4. write a program to find the area of a triangle

Hint : area = b\*h/2

b = int(input('Enter breadth value'))

h = int(input('Enter height value'))

area = b\*h/2

print('Area of a triangle = ',area)

5. write a program to find the area of a circle

Hint : area = pi r

r = int(input('Enter radius value'))

area = 3.14 \* r \* r

print('Area of a circle = ',area)

6. write a program to find the square of a given number

n = int(input('Enter any number'))

s = n \* n

print('Square of a given number = ',s)

7. write a program to find the cube of a given number

n = int(input('Enter any number'))

c = n \* n \* n

print('Cube of a given number = ',c)

8. write a program to read the student hall ticket number,

name of the student and three subject marks .

To find the total and average.

htno = int(input('Enter hall ticket number'))

name = input('Enter student Name :')

m1 = int(input('Enter First subject Marks:'))

m2 = int(input('Enter Second subject Marks:'))

m3 = int(input('Enter Third subject Marks:'))

total = m1+m2+m3

avg = total/3

print('\n\n')

print('\t\t\t--------------')

print('\t\t\tSTUDENT RESULT')

print('\t\t\t--------------')

print('\t\t\tTotal Marks = ',total)

print('\t\t\tAverage Marks = ',avg)

**CONTROL FLOW**

**or**

**control structures**

**or**

**control statements**

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

CONTROL FLOW

|

|

|

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

| | |

conditional Iterative Transfer

control control control

statements statements statements

| | |

|--> if |--> while |--> break

|--> if-else |--> for |--> continue

|--> if-elif |--> pass

|--> if-elif-else

|--> nested if statement

|--> single line if statement

|--> match case statement

**conditional control statements :**

==> These statements are executed based on condition.

1. if statement

2. if-else statement

3. if-elif statement

4. if-elif-else statement

5. nested if statement

6. single line if statement

7. match case statement

1. if statement

2. if-else statement:

==> syntax:

if condition:

st-1

else:

st-2

Here,

==> if,else are keywords.

==> condition is a relational expression.

It returns either True/False.

==> if the given condition returns True, then,

st-1 is executed and st-2 is not executed.

==> if the given condition returns False, then,

st-1 is not executed and st-2 is executed.

write a program you are elegible for blood donation or not ?

weight = int(input('Enter your weight'))

if weight>=45:

print('you are elegible for blood donation')

else:

print('you are not elegible for blood donation')

2. write a program to find the biggest of two numbers

a = int(input('Enter First number'))

b = int(input('Enter second number'))

if a>b:

print('A is biggest')

else:

print('B is biggest')

3. write a program to find the given number is even or odd

n = int(input('Enter any number'))

if n%2==0:

print('The given number is even number ')

else:

print('The given number is odd number ')

4. write a program to find you are elegible for voting or not

age = int(input('Enter your age'))

if age>=18:

print('congrats ....')

print('you are elegible for voting ')

print('please participate present elections ...')

else:

print('sorry ..')

print('you are not elegible for voting')

print('please try, next elections ....')

'''

booking

counter

|

|

True | False

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

| |

| |

| |

cinema restarent

| |

| |

|-----------home------------|

'''

1. if statement:

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

==> It is also called simple if statement.

==> syntax:

if condition:

st-1

Here,

==> if is a keyword.

==> if the given condition returns True, then,

st-1 is executed.

==> if the given condition returns False, then,

st-1 is not executed.

ex-1: write a program to find the biggest of two numbers '

a = int(input('Enter First number'))

b = int(input('Enter second number'))

if a>b:

print('A is biggest')

3. if-elif statement:

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

==> This statement is used to check multiple conditions.

==> syntax:

if condition-1:

st-1

elif condition-2:

st-2

elif condition-3:

st-3

---

---

elif condition-n:

st-n

Here,

==> if,elif are keywords.

==> condition-1 returns True, then, st-1 is executed.

==> condition-1 returns False, then, go to condition-2

==> condition-2 returns True, then, st-2 is executed.

==> condition-2 returns False, then, go to condition-3

==> condition-3 returns True, then, st-3 is executed.

==> condition-3 returns False, then, go to condition-4,

......etc.

==> condition-n returns True, then, st-n is executed.

==> condition-n returns False, then, do nothing

ex-1: write a program to find the biggest of two numbers

a = int(input('Enter First number'))

b = int(input('Enter second number'))

if a>b:

print('A is biggest')

elif b>a:

print('B is biggest')

elif a==b:

print('A & B are equal')

ex-2: write a program to find the given number

is positive number/negative number/zero number

n = int(input('Enter any number'))

if n>0:

print('Given Number is positive number')

elif n<0:

print('Given Number is negative number')

elif n==0:

print('Given Number is zero number')

4. if-elif-else statement:

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

==> This statement is used to check multiple conditions

with else block.

==> syntax:

if condition:

st-1

elif condition:

st-2

elif condition:

st-3

---

---

elif condition:

st-n

else:

else block

ex-1:

write a program to enter day number and print it words

daynum = int(input('Enter day number'))

if daynum==0:

print('SUNDAY')

elif daynum==1:

print('MONDAY')

elif daynum==2 :

print('TUESDAY')

elif daynum== 3:

print('WEDNESDAY')

elif daynum==4 :

print('THURSDAY')

elif daynum==5 :

print('FRIDAY')

elif daynum== 6:

print('SATURDAY')

else:

print('wrong input,pls enter correct input ....')

ex-2:

write a program to enter month number and print it words

monnum = int(input('Enter month number'))

if monnum==1:

print('JANUARY')

elif monnum==2 :

print(' ')

elif == 3:

print(' ')

elif ==4 :

print(' ')

elif ==5 :

print(' ')

elif == 6:

print(' ')

elif monnum==7 :

print(' ')

elif == 8:

print(' ')

elif ==9 :

print(' ')

elif ==10 :

print(' ')

elif == 11:

print(' ')

elif == 12:

print(' ')

else:

print('wrong Month number ,pls enter correct input ....')

write a program to enter any single digit number

and print it words

n = int(input('Enter any single digit number(0 to 9) '))

if n==0:

print('ZERO')

elif n==1:

print('ONE')

elif n==2 :

print('TWO')

elif n== 3:

print('THREE')

elif n==4 :

print('FOUR')

elif n==5 :

print('FIVE')

elif n== 6:

print('SIX')

elif n== 7:

print('SEVEN')

elif n== 8:

print('EIGHT')

elif n== 9:

print('NINE')

else:

print('wrong input,pls enter correct input ....')

5. nested if statement:

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

==> A if statement within another if statement.

It is called nested if statment.

==> syntax-1:

if condition-1:

if condition-2:

st-1

Here,

==> if the condition-1 returns True, and condition-2 returns True.

then, st-1 is executed.

==> syntax-2:

if condition-1:

if condition-2:

st-1

else:

st-2

else:

st-3

ex-1: write a program to find,

you are elegible for blood donation or not ?

age = int(input('Enter your age'))

if age>=18:

weight = int(input('Enter your weight'))

if weight>=45:

print('you are elegible for blood donation')

else:

print('you are not elegible for blood donation')

print('under weight')

else:

print('you are not elegible for blood donation')

print('under Age')

ex-2: write a program to find the given number

positive number/negative/zero number

n = int(input('Enter any number'))

if n>=0:

if n>0:

print('Given number is positive number')

else:

print('Given number is zero number')

else:

print('Given number is negative number')

6. single line if statement:

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

==> A if statement written in only one line.

It is called single line if statement.

==> syntax:

st-1 if condition else st-2

Here,

==> if,else are keywords.

==> if the given condition returns True, then,

st-1 is executed.

==> if the given condition returns False, then,

st-2 is executed.

ex-1: write a program you are elegible for voting or not ?

method-1:

age = int(input('Enter your age'))

print('Elegible') if age>=18 else print('Not Elegible')

method-2:

age = int(input('Enter your age'))

print('Elegible' if age>=18 else 'Not Elegible')

ex-2: write a program to find the given number is even/odd?

n = int(input('Enter any number'))

print('Even Number') if n%2==0 else print('Odd number')

ex-3: write a program to find the given number is even/odd?

n = int(input('Enter any number'))

print('positive') if n>0 else print('zero') if n==0 else print('negative')

**Iterative control statements**

**or**

**Repetative control statements:**

**or**

**Loops**

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

what is loop?

==> A single statement or group of statements

are executed repeatedly.

It is called loop.

===> python supports 2 types of loops:

1. while

2. for

1. while loop:

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

==> A single statement or group of statements

are executed repeatedly until condition returns True.

==> if the given condition returns False, then, to stop the loop.

==> syntax:

while condition:

st-1

st-2

st-3

----

----

st-n

Here,

while is a keyword

condition returns True/False.

':' begining of the while loop

st-1,st-2,...st-n are called body of the while loop.

explanation:

step-1. To check the condition.

step-2: if the condtioin returns True, then,

step-3: to execute body of the while loop.

step-4: Repeate step-1 to step-3,

until condition returns True.

step-5: if the condtiion returns False, then, to stop the loop.

ex-1: To print good morning 10 times

method-1: using simple print() statement

print('Good Morning')

print('Good Morning')

print('Good Morning')

print('Good Morning')

print('Good Morning')

print('Good Morning')

print('Good Morning')

print('Good Morning')

print('Good Morning')

print('Good Morning')

method-2: using repetition operator

print('Good Morning\n'\*10)

method-3: using while loop

i=1

while i<=10:

print('Good Morning')

i=i+1

Here,

i=1 is called initialization of a given variable

i<=10 is called condition.

i=i+1 is called increment of the given variable.

ex-2:

while i<=10:

print('Good Morning')

i=i+1

Note:

It returns Error, becuase 'i' is not defined.

ex-3:

i=1

while :

print('Good Morning')

i=i+1

Note: It returns syntax Error

ex-4:

i=1

while i<=10:

print('Good Morning')

Note: The above statement is executed repeatedly

infinite number of times.

ex-5: write a program To print 1 to 10 numbers

i=1

while i<=10:

print(i)

i=i+1

ex-6: write a program To print odd numbers between 1 to 10

i=1

while i<=10:

print(i)

i=i+2

ex-7: write a program To print even numbers between 1 to 10

i=2

while i<=10:

print(i)

i=i+2

ex-8: write a program To print 10 to 1 numbers

i=10

while i>=1:

print(i)

i=i-1

range():

==> It is a built-in function.

==> This function returns group of values(sequences/series).

==> syntax:

range(start,stop,step)

Here,

==> start indicates starting value.

It is optional. By default starting value is '0'.

==> stop indicates ending value.

It is compulsory.

==> step indicates either increment/decrement.

It is optional. By default, the value increased by '1'.

ex:

1. range(1,11,1) ---> 1,2,3,4,5,6,7,8,9,10

2. range(1,10,2) --> 1, 3, 5, 7 ,9

3. range(2,11,2) --> 2, 4, 6 ,8, 10

4. range(5,31,5) --> 5 10 15 20 25 30

5. range(10,61,10) --> 10 20 30 40 50 60

6. range(10,0,-1) --> 10 9 8 7 6 5 4 3 2 1

7. range(1,6,1) --> 1,2,3,4,5

8. range(1,6) --> 1,2,3,4,5

9. range(6) --> 0,1,2,3,4,5

10. range(1) --> 0

11. range(0) --> no values

12. range() --> Error

13. range(1,1,1) --> no values

14. range(-1,-1,-1) --> no values

15. range(0,-1,-1) --> 0

for loop:

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

==> The while loop is executed based on condition.

==> if we want to executed 'n' number of times then,

we go for loop.

==> syntax:

for variablename in sequence:

st-1

st-2

st-3

--

--

st-n

Here,

==> for, is are keywords

==> variablename is a control variable.

==> sequence indicates group of values.

It may be either list/tuple/set/dict/range/string

==> ':' begining of the for loop.

==> st-1,st-2,....st-n are called body of the for loop.

ex-1: To print 1 to 10 numbers

for i in range(1,11,1):

print(i)

ex-2: To print odd numbers between 1 to 10 numbers

for i in range(1,10,2):

print(i)

ex-3: To print even numbers between 1 to 10 numbers

for i in range(2,11,2):

print(i)

ex-4:

for i in range(5,31,5):

print(i)

ex-5 ...............................ex-15

ex-16:

for i in 'tronix':

print(i)

ex-17:

s = 'tronix'

for i in s:

print(i)

ex-18:

x = [11,22,33,44,55,66]

for i in x:

print(i)

nested Loops:

==> A loop within another loop.

this process is called nested loop.

==> python supports 4 types of nested loops:

1. nested while loop

2. nested for loop

3. for loop inside while loop

4. while loop inside for loop

**1. nested while loop:**

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

==> A while loop inside another while loop.

It is called nested while loop.

==> syntax:

while conndition:

while condition:

st-1

st-2

st-3

--

--

st-n

ex-1:

i=1

while i<=5:

j=1

while j<=5:

print('I = ', i,',J = ',j)

j=j+1

i=i+1

2. nested for loop:

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

==> A for loop within another for loop.

This process is called nested for loop.

==> syntax:

for variablename in sequence:

for variablename in sequence :

st-1

st-2

st-3

--

--

st-n

ex-1:

for i in range(1,6,1):

for j in range(1,6,1):

print('I = ', i,',J = ',j)

3. for loop inside while loop:

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

while condition:

for variablename in sequence :

st-1

st-2

st-3

--

--

st-n

ex-1:

i=1

while i<=5:

for j in range(1,6,1):

print('I = ', i,',J = ',j)

i=i+1

4. while loop inside for loop:

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

for variablename in sequence :

while condition:

st-1

st-2

st-3

--

--

st-n

ex-1:

for i in range(1,6,1):

j=1

while j<=5:

print('I = ', i,',J = ',j)

j=j+1

write a program to print a given mathematical table

9 x 1 = 9

9 x 2 = 18

9 x 3 = 27

---

---

9 x 10 = 90

n = int(input('Enter Table Number :'))

for i in range(1,11,1):

print(n,'\*',i,'=',n\*i)

write a program to print 1 to 20 tables

for n in range(1,21,1):

print('\n\nTable No : ',n)

print('--------------')

for i in range(1,11,1):

print(n,'\*',i,'=',n\*i)

x = 'tronix'

for i in x:

print(i)

print('good bye')

indentation is a fundamental principle in python,

which you will encounter many times in the future.

Generally, it translaters into 4 blank spaces or a '1' tab key.

write a program to find the No. of digits in a given number.

ex-1: 467 --> NO. of digits = 3

ex-2: 4895348 --> NO. of digits = 7

n = input('Enter any number') --> 4895348

count = 0

for i in n:

count = count + 1

print('No. of digits = ',count)

write a program sum of 1 to 10 numbers

sum = 0

for i in range(1,11,1):

sum = sum + i

print('sum of 1 to 10 numbers = ',sum)

write a program sum of 1 to 10 numbers

r1 = int(input('Enter starting point'))

r2 = int(input('Enter ending point'))

sum = 0

for i in range(r1,r2+1,1):

sum = sum + i

print('sum of ',r1, 'to', r2 , 'numbers = ',sum)

Transfer control statements:

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

==> To move the cursor from,

one part of the program into another part of the program.

==> python supports 3 types of transfer control statements:

1. break

2. continue

3. pass

1. break statement:

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

==> break is a keyword.

==> This statement is used to exit from loop.

==> It is used either while loop or for loop.

==> It is also used for nested loops.

==> syntax:

break

ex-1 : break statement usig while loop

i=1

while i<=10:

if i==7:

break

else:

print(i)

i=i+1

ex-2 : break statement usig for loop

for i in range(1,11,1):

if i==7:

break

else:

print(i)

ex-3:

s = 'python'

for i in s:

if i=='h':

break

else:

print(i)

ex-4:

x = [11,22,33,44,55,66,77,88,99,100]

for i in x:

if i==66:

break

else:

print(i)

ex-5:

while True:

print('Hello...')

The above loop repeated infinite number of times

ex-6:

while True:

print('Hello...')

ch = input('Do u want to continue(y/n)?')

if ch=='n':

break

ex-7:

while True:

name = input('Enter your name')

ch = input('Do u want to continue(y/n)?')

if ch=='n':

break

2. continue statement:

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

==> continue is a keyword.

==> This statement is used to,

transfer the control begining of the loop.

==> It is used either while loop or for loop.

==> It is also used for nested loops.

==> syntax:

continue

ex-1 :

for i in range(1,11,1):

if i==7:

continue

else:

print(i)

ex-2 :

s = 'python'

for i in s:

if i=='h':

continue

else:

print(i)

ex-3:

x = [11,22,33,44,55,66,77,88,99,100]

for i in x:

if i==66:

continue

else:

print(i)

ex-4:

x = [11,22,66,33,44,55,66,77,88,99,66,56,66,100]

for i in x:

if i==66:

continue

else:

print(i)

ex-5:

s = 'python is high level programming language'

for i in s:

if i=='h':

continue

else:

print(i)

3. pass statement:

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

==> pass is a keyword.

==> This keyword is used to create a empty statement.

==> It is used to create empty if statement, or

empty while loop or empty for loop or

empty function or empty class.

==> syntax:

pass

ex-1:

a = 5679

b = 38

if a>b:

pass

else:

print('B is biggest')

ex-2:

for i in range(1,11,1):

pass

ex-3:

while True:

pass

write a program to perform arithmetic operations '''

while True:

print('\t\tArithmetic operations Menu')

print('\t\t---------------------------')

print('\t\t1. Addition')

print('\t\t2. subtraction')

print('\t\t3. Multiplication')

print('\t\t4. division')

print('\t\t5. Exit')

op = int(input('Enter which arithmetic operation u want '))

if op == 5:

break

a = int(input('Enter first number'))

b = int(input('Enter second number'))

if op==1:

c=a+b

print(c)

elif op==2:

c=a-b

print(c)

elif op==3:

c=a\*b

print(c)

elif op==4:

c=a/b

print(c)

1. write a progarm to find the sum of series

1 + 2 + 3 + 4 + .....n

n = int(input('Enter any number'))

sum = 0

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

sum = sum + i

print('sum value = ',sum)

2. write a progarm to find the sum of series

1 + 2 + 3 + 4 + .....n

n = int(input('Enter any number'))

sum = 0

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

sum = sum + i\*i

print('sum value = ',sum)

**match case (switch statement):**

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

==> It is used to check multiple conditions.

==> it is a new feature.

==> It is availabe 3.10 onwards.

==> syntax:

match expression:

case exp-1:

st-1

case exp-2:

st-2

case exp-3:

st-3

---

---

case exp-n:

st-n

case \_:

default statements

ex-1: write a program to enter day number

and print it words

daynum = int(input('Enter day number'))

match daynum:

case 0:

print('SUNDAY')

case 1:

print('MONDAY')

case 2:

print('TUESDAY')

case 3:

print('WEDNESDAY')

case 4:

print('THURSDAY')

case 5:

print('FRIDAY')

case 6:

print('SATURDAY')

case \_:

print('Invalid input')

ex-2: write a program to enter month number

and print it words

monnum = int(input('Enter month number'))

match monnum:

case 1:

print('JANUARY')

case 2:

print('FEBRUARY')

case 3:

print('MARCH')

case 4:

print('APRIL')

case 5:

print('MAY')

case 6:

print('JUNE')

case 7:

print('JULY')

case 8:

print('AUGUST')

case 9:

print('SEPTEMBER')

case 10:

print('OCTOBER')

case 11:

print('NOVEMBER')

case 12:

print('DECEMBER')

case \_:

print('Invalid month number, please enter correct month number' )

ex-3: write a program enter two numbers and to perform

arithmetic operations

a = int(input('Enter first number'))

b = int(input('Enter second number'))

op = input('which arithmetic operation u want ?')

match op:

case 'add':

c = a+b

print(c)

case 'sub':

c = a-b

print(c)

case 'mul':

c = a\*b

print(c)

case 'div':

c=a/b

print(c)

case 'mod':

c=a%b

print(c)

case \_:

print('Invalid input ....pls enter correct input')

ex-4: write a program to enter single digit number

and print it words

1. write a program to find the factors of a given number.

n = int(input('Enter any number'))

i=1

print('The Factors Are :')

while i<=n:

if n%i == 0:

print(i)

i=i+1

2. write a program to count No. of factors of a given number

n = int(input('Enter any number'))

count = 0

i=1

print('The Factors Are :')

while i<=n:

if n%i == 0:

print(i)

count = count + 1

i=i+1

print('No. of factors = ',count)

**data structures**

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

data --> only values

structure ---> shape

data structure:

The way of organizing to storing the

data is called data structure.

storage:

ordinary variable:

==> it is a variable

==> It is used to store only one value at a time.

ex-1:

a = 35

--

--

a = 10

--

--

a = 75

--

--

a = 12

--

--

a = 3

array variable:

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

==> It is a variable.

==> It is used to store morethan one value.

==> All the values must be same data type.

ex-1:

int a[5] = {10,75,12,34,3}

float x[3] = { 1.3,9,5,6.1234} }

--

a = 3

==> if we want store morethan one value with different data types.

then, we go for python data structures.

==> python supports 5 types of data structures:

1. list data structures

2. tuple data structures

3. set data structures

4. dict data structures

5. string data structures

**LIST DATA STRUCTURE**

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

properties(characterstics):

1. A group of values, each value separated by commas.

and enclosed by square brackets. It is called a list.

2. lists are created by using list().

3. All elements(values) are stored orderly.

4. Insertions are allowed.

5. Deletions are allowed.

6. List supports duplicate elements.

7. list supports indexing and slicing.

8. list is mutable.

9. list is dynamic

10. list supports heterogeneous elements.

11. list declaration syntax:

listname = [value-1,value-2,value-4,..... value-n ]

ex-1: To create a empty list using square brackets.

a = [ ]

print(a)

print(type(a))

ex-2: To create a empty list using list() function.

a = list()

print(a)

print(type(a))

ex-3: To create a list with elements

a = [10,20,30,40,50]

print(a)

print(type(a))

ex-4: To create a list with duplicate elements

a = [10,20,10,10,10,30,40,40,50,30,30,40]

print(a)

print(type(a))

Elements are 2 types:

1. homogeneous elements

2. heterogeneous elements

1. homogeneous elements:

==> All the elements are same data type.

It is called homogeneous elements.

ex: boys hostel,girls hostel,girls junior college,....etc.

2. heterogeneous elements:

==> All the elements are different data types.

It is called heterogeneous elements.

ex: cinema hall,class room, shopping mall ....etc.

ex-5: To create a list with homogeneous elements

a = [11,22,33,44,55,66]

print(a)

print(type(a))

ex-6: To create a list with heterogeneous elements

a = [1901,'ram kumar',77,88,99,88.00,'pass', 'A + Grade' ]

print(a)

print(type(a))

'''

ex-7: To create a list using list()

a = list(range(1,11,1))

print(a)

print(type(a))

Accessing List elements using indexing:

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

a). Every element has some position in the list,

known as the index.

b). The list is made up of multiple parts.

And each section of the list is an element.

c). We can access all the values by specifying the

corresponding integer index.

d). python supports positive and negative indexing.

==> positive index starts from left to right.

i.e. 0,1,2,3,....size-1

==> negative index starts from right to left.

i.e. -1,-2,-3,-4,....-size.

ex-1: To print list elements using positive indexing

a = [11,22,33,44,55]

print(a)

print(a[0])

print(a[1])

print(a[2])

print(a[3])

print(a[4])

print(a[5])

ex-2: To print list elements using negative indexing

a = [11,22,33,44,55]

print(a[-1])

print(a[-2])

print(a[-3])

print(a[-4])

print(a[-5])

Accessing list elements using slicing:

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

==> indexing is used to access only one element.

==> if we want to access morethan one element,

we go for slicing.

==> syntax:

[start:stop:step]

Here,

==> ':' is called slice operator.

==> start indicates starting index number

It is optional. By default starting index number is '0'.

==> stop indicates ending index number

It is optional.

By default ending index number is end of the list.

==> step indicates either increment/decrement of the index number

It is optional. By default increased by '1'.

nindex -14 -13 -12-11-10-9 -8 -7 -6 -5 -4 -3 -2 -1

listname = 11 22 33 44 55 66 77 88 99 100 200 300 400 500

pindex 0 1 2 3 4 5 6 7 8 9 10 11 12 13

ex-1:

a = [11,22,33,44,55,66,77,88,99,100,200,300,400,500]

#print(a[4:12:1])

#print(a[2:6:1])

#print(a[0:9:1])

#print(a[ :5:1])

#print(a[ : :1])

#print(a[ : : ])

#print(a[ : :3])

print(a[-4 : -12 :-1])

Mutable:

==> The values can be changed during execution of the program.

It is called mutable.

ex: age,bank balance,weather,temparature,time,date,month,

life style,weight,government, ...etc.

Immutable:

==> The values cannot be changed during execution of the program.

It is called Immutable.

ex: name,surname,color,....etc.

list Vs. mutable:

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

==> once we create a list, the values can be changed

during execution the program

==> It is called list mutable

ex-1:

a = [11,22,33,44,55]

print(a)

a[0]=100

a[-1]=99

print(a)

**built-in functions in lists or methods in lists:**

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

1. len() -🡪 no. of elements in list

2. max()🡪 maximum element in list

3. min()🡪minimum element in list

4. sum()🡪sum of elements in list

5. append()🡪 adding the elements at end of the list

6. insert()🡪adding the element at given index number in list

7. extend()🡪adding element at end of the list

8. pop()🡪removing the element at end of the list

9. remove()🡪remove particular element in the list

10. clear()🡪remove the all elements

11. copy()🡪copy the elements

12. reverse()🡪the arrangement of the elements to be reversed

13. sort()🡪sort the elements in assending order/decending order

14. count()🡪 it shows how many times the element is present in list

15. index()🡪it shows the element by using index number

1. len():

==> This function is used to find the No. of elements

present in a list.

2. max():

==> This function is used to find the highest value in a list.

3. min():

==> This function is used to find the lowest value in a list.

4. sum():

==> This function is used to find the sum of list elements

ex-1 :

a = [11,3,7,99,23,5]

print('No. of elements = ',len(a))

print('Highest value = ',max(a))

print('Lowest value = ',min(a))

print('sum of List elements = ',sum(a))

Inserting new elements into a list:

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

5. append():

==> This function is used to add new element end of the list.

==> It is used to add only one element.

ex-1:

a=[11,22,33,44]

print(a)

a.append(99)

a.append(8😎

a.append(77)

a.append(66)

print(a)

6. insert():

==> This function is used to add new element at given index number.

==> It is used to add only one element.

==> syntax:

insert(indexnumber,new value)

ex-1:

a=[11,22,33,44]

print(a)

a.insert(1,99)

a.insert(0,66)

a.insert(-1,8😎

print(a)

7. extend():

==> This function is used to add new elements end of the list.

==> It is used to add morethan one element.

ex-1:

a=[11,22,33,44]

print(a)

a.extend([99,88,77,66,55])

print(a)

ex-2:

a=[11,22,33,44]

b=[99,88,77,66,55]

print(a)

a.extend(b)

print(a)

ex-3:

a=[11,22,33,44]

print(a)

a.extend(range(1,11,1))

print(a)

Removing elements from a list:

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

**8. pop():**

==> This function is used to remove the last element from a list.

==> It is also used for remove the given index number

value from a list.

==> syntax:

pop(indexnumber)

==> if the index number is present, remove it.

==> if the index number is not present, then, it returns

IndexError.

ex-1:

a=[11,22,33,44,55]

print(a)

a.pop()

a.pop()

a.pop()

a.pop()

a.pop()

print(a)

a.pop()

print(a)

ex-2:

a=[11,22,33,44,55]

print(a)

a.pop(3)

a.pop(0)

a.pop(-1)

print(a)

ex-3:

a=[11,22,33,44,55]

print(a)

a.pop(9)

print(a)

**9. remove():**

==> This function is used to remove the given element from a list.

==> if the given element is not present, in a list,

then, it returns ValueError.

ex-1:

a=[11,22,33,44,55]

print(a)

a.remove(33)

a.remove(11)

a.remove(55)

print(a)

ex-2:

a=[11,22,33,44,55]

print(a)

a.remove(363)

a.remove(11)

a.remove(55)

print(a)

**10. clear():**

==> This function is used to remove the all elements from a list.

ex-1:

a = [11,22,33,44,55]

print(a)

a.clear()

print(a)

**11.copy():**

==> This function is used to copy the elements

from one list into another list.

ex-1:

a = [11,22,33,44,55]

b = a.copy()

print(a)

print(b)

**12. reverse():**

==> This function is used to arrange the elements

reverse order.

==> The result is assigned in same list.

ex-1:

a = [11,22,33,44,55]

print(a)

a.reverse(33)

print(a)

**13. sort():**

==> sorting order means to arrange elements

either ascending order or descending order

==> ascending order means to arrange elements lowest to highest

==> descending order means to arrange elements highest to lowest

==> by default , sorting order is ascending order.

ex-1: To arrange elements ascending order

a = [ 9,78,12,3,67 ]

print(a)

a.sort()

print(a)

ex-2: To arrange elements descending order '

method-1:

a = [ 9,78,12,3,67 ]

print(a)

a.sort()

a.reverse()

print(a)

method-2:

a = [ 9,78,12,3,67 ]

print(a)

a.sort(reverse=True)

print(a)

**14. count():**

==> This function is used to find, how many times

a given element present in a list.

==> if the given element, not present, in a list, the,

it returns zero

ex-1:

a = [11,22,11,11,55,44,66,99,66,44,44,44,44]

print(a.count(55))

print(a.count(11))

print(a.count(44))

print(a.count(99))

print(a.count(88))

print(a.count(77))

**15. index():**

==> This function returns, index number of the given value.

==> if the given element, not present, in a list, the,

it returns ValueError

ex-1:

a = [11,22,11,11,55,44,66,99,66,44,44,44,44]

print(a.index(55))

print(a.index(11))

print(a.index(44))

print(a.index(99))

print(a.index(88))

print(a.index(77))

arithmetic operators in lists:

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

1. list concatenation operator(+)

2. list repetition operator(\*)

1. list concatenation operator(+):

==> This operator is used to concatenation(join or merging)

of 2 or more lists into another list.

ex-1 :

a = [11,22,33,44,55]

b = [99,88,77,66]

c = ['ram','sita','venkat']

d= a+b+c

print(d)

2. list repetition operator(\*):

==> This operator is used to

repeat a list 'n' number of times.

ex-1 :

a = [11,22,33,44,55]

b = [99,88,77,66]

c = ['ram','sita','venkat']

print(a\*3)

print(b\*10)

print(c\*5)

membership operators in lists:

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

==> The member ship operators are used for , find the

given value is present or not in the list.

in : if the value is present in a list, then,

it returns True. otherwise it returns False.

not in : if the value is not present in a list, then,

it returns True. otherwise it returns False.

ex-1:

a=[11,22,33,44,55]

print(33 in a )

print(99 in a)

print(22 not in a )

print(88 not in a)

**TUPLE DATA STRUCTURE**

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

properties(characterstics):

1. A group of values, each value separated by commas.

and enclosed by parenthesis. It is called a tuple.

2. Tuples are created by using tuple().

3. All elements(values) are stored orderly.

4. Insertions are not allowed.

5. Deletions are not allowed.

6. tuple supports duplicate elements.

7. tuple supports indexing and slicing.

8. tuple is Immutable.

9. tuple is static.

10. tuple supports heterogeneous elements.

11. tuple declaration syntax:

tuplename = ( value-1,value-2,value-3,..... value-n )

12. In tuples, parenthesis are optional.

ex-1: To create a empty tuple using parenthesis.

t = ( )

print(t)

print(type(t))

ex-2: To create a empty list using list() function.

t = tuple()

print(t)

print(type(t))

ex-3: To create a tuple with elements

t = ( 10,20,30,40,50)

print(t)

print(type(t))

ex-4: To create a tuple with duplicate elements

t = ( 10,20,10,10,10,30,40,40,50,30,30,40 )

print(t)

print(type(t))

Elements are 2 types:

1. homogeneous elements

2. heterogeneous elements

1. homogeneous elements:

==> All the elements are same data type.

It is called homogeneous elements.

ex: boys hostel,girls hostel,girls junior college,....etc.

2. heterogeneous elements:

==> All the elements are different data types.

It is called heterogeneous elements.

ex: cinema hall,class room, shopping mall ....etc.

ex-5: To create a tuple with homogeneous elements

t = ( 11,22,33,44,55,66 )

print(t)

print(type(t))

ex-6: To create a tuple with heterogeneous elements

t = ( 1901,'ram kumar',77,88,99,88.00,'pass', 'A + Grade' )

print(t)

print(type(t))

ex-7: To create a tuple using tuple() & range()

t = tuple(range(1,11,1))

print(t)

print(type(t))

Here,

range(1,11,1) ---> It returns 1 to 10 numbers

tuple() --> These numbers convert into tuple data type.

ex-8: To create a tuple without parenthesis

x = 11,22,33,44,55

print(x)

print(type(x))

ex-9: To create a single valued tuple with comma

x = 11,

print(x)

print(type(x))

Accessing tuple elements using indexing:

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

a). Every element has some position in the tuple,

known as the index.

b). The tuple is made up of multiple parts.

And each section of the tuple is an element.

c). We can access all the values by specifying the

corresponding integer index.

d). python supports positive and negative indexing.

==> positive index starts from left to right.

i.e. 0,1,2,3,....size-1

==> negative index starts from right to left.

i.e. -1,-2,-3,-4,....-size.

ex-1: To print tuple elements using positive indexing

a = ( 11,22,33,44,55 )

print(a)

print(a[0])

print(a[1])

print(a[2])

print(a[3])

print(a[4])

print(a[5])

ex-2: To print tuple elements using negative indexing

a = [11,22,33,44,55]

print(a[-1])

print(a[-2])

print(a[-3])

print(a[-4])

print(a[-5])

Accessing tuple elements using slicing:

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

==> indexing is used to access only one element.

==> if we want to access morethan one element,

we go for slicing.

==> syntax:

[start:stop:step]

Here,

==> ':' is called slice operator.

==> start indicates starting index number

It is optional. By default starting index number is '0'.

==> stop indicates ending index number

It is optional.

By default ending index number is end of the list.

==> step indicates either increment/decrement of the index number

It is optional. By default increased by '1'.

nindex -14 -13 -12-11-10-9 -8 -7 -6 -5 -4 -3 -2 -1

listname = 11 22 33 44 55 66 77 88 99 100 200 300 400 500

pindex 0 1 2 3 4 5 6 7 8 9 10 11 12 13

ex-1:

a = ( 11,22,33,44,55,66,77,88,99,100,200,300,400,500)

#print(a[4:12:1])

#print(a[2:6:1])

#print(a[0:9:1])

#print(a[ :5:1])

#print(a[ : :1])

#print(a[ : : ])

#print(a[ : :3])

print(a[-4 : -12 :-1])

Mutable:

==> The values can be changed during execution of the program.

It is called mutable.

ex: age,bank balance,weather,temparature,time,date,month,

life style,weight,government, ...etc.

Immutable:

==> The values cannot be changed during execution of the program.

It is called Immutable.

ex: name,surname,color,....etc.

**Tuple Vs. Immutable**:

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

==> once we create a tuple, the values cannot be changed

during execution of the program

==> It is called Tuple Immutable

ex-1:

t = ( 11,22,33,44,55 )

print(t)

t[0]=100

t[-1]=99

print(t)

Note:

t[0]=100

TypeError: 'tuple' object does not support item assignment

built-in functions in tuples or methods in tuples:

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

1. len()

2. max()

3. min()

4. sum()

5. count()

6.. index()

**1. len():**

==> This function is used to find the No. of elements

present in a tuple.

2. max():

==> This function is used to find the highest value in a tuple.

3. min():

==> This function is used to find the lowest value in a tuple.

4. sum():

==> This function is used to find the sum of tuple elements

ex-1 :

t = ( 11,3,7,99,23,5)

print('No. of elements = ',len(t))

print('Highest value = ',max(t))

print('Lowest value = ',min(t))

print('sum of tuple elements = ',sum(t))

**5. count():**

==> This function is used to find, how many times

a given element present in a tuple.

==> if the given element, not present, in a tuple, the,

it returns zero

ex-1:

a = ( 11,22,11,11,55,44,66,99,66,44,44,44,44 )

print(a.count(55))

print(a.count(11))

print(a.count(44))

print(a.count(99))

print(a.count(88))

print(a.count(77))

**6. index():**

==> This function returns, index number of the given value.

==> if the given element, not present, in a tuple, then,

it returns ValueError

ex-1:

a = (11,22,11,11,55,44,66,99,66,44,44,44,44 )

print(a.index(55))

print(a.index(11))

print(a.index(44))

print(a.index(99))

print(a.index(88))

print(a.index(77))

**arithmetic operators in tuples:**

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

1. tuple concatenation operator(+)

2. tuple repetition operator(\*)

1. tuple concatenation operator(+):

==> This operator is used to concatenation(join or merging)

of 2 or more tuples into another tuple.

ex-1 :

a = (11,22,33,44,55)

b = (99,88,77,66)

c = ('ram','sita','venkat')

d= a+b+c

print(d)

print(type(d))

ex-2:

t1 = ( )

t2 = ( )

t3 = t1+t2

print(t3)

print(type(t3))

2. tuple repetition operator(\*):

==> This operator is used to

repeat a tuple 'n' number of times.

ex-1 :

a = (11,22,33,44,55)

b = (99,88,77,66)

c = ('ram','sita','venkat')

print(a\*3)

print(b\*10)

print(c\*5)

**membership operators in tuples:**

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

==> The membership operators are used for , find the

given value is present or not in the tuple.

in : if the value is present in a tuple, then,

it returns True. otherwise it returns False.

not in : if the value is not present in a tuple, then,

it returns True. otherwise it returns False.

ex-1:

t= (11,22,33,44,55 )

print(33 in t )

print(99 in t)

print(22 not in t )

print(88 not in t)

**SET DATA STRUCTURE**

properties(characterstics):

1. A group of values, each value separated by commas

and enclosed by curly braces. It is called set.

2. sets are created by using set()

3. All elements are stored unorderly.

i.e. A group of elements are stored unorderly.

4. Insertions are allowed

5. Deletions are alllowed

6. Duplicate elements are not allowed.

7. set does not supports indexing and slicing

8. set is mutable

9. set is dynamic.

Based on our requirement the size can be either

increased or decreased.

10. set supports heterogeneous elements

11. set declaration syntax is:

setname = { value-1,value-2,value-3,....value-n }

ex-1: To create a empty set using curly braces

s = { }

print(s)

print(type(s))

Here,

s is dict type

Note: empty sets are created by using set() function only

ex-2: To create a empty set using set()

s = set()

print(s)

print(type(s))

ex-3: To create a set with elements

s ={11,22,33,44,55}

print(s)

print(type(s))

ex-4: To create a set using set() & range() functions

s = set(range(1,11,1))

print(s)

print(type(s))

Note:

range() --> It returns 1 to 10 numbers

set() --> To convert these numbers into set data type.

ex-5: To create a set with homogeneous elements

s = {8,67,34,12,88}

print(s)

print(type(s))

ex-6: To create a set with heterogeneous elements

s = {8,67.50,'hello',12.00,88,7+4j,True}

print(s)

print(type(s))

ex-7: To create a set with duplicate elements

s = {8,11,23,11,11,11,11,11,8,8,8 }

print(s)

print(type(s))

print(len(s))

Built-in functions or methods in sets:

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

1. len()

2. max()

3. min()

4. sum()

5. add()

6. update()

7. pop()

8. remove()

9. discard()

10. clear()

11. copy()

**mathematical related functions in sets:**

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

1. set union()

2. set intersection()

3. set difference()

4. set symmetric\_difference()

5. issubset()

6. issuperset()

7. isdisjoint()

**1. len():**

It returns NO. of elements present in a set.

**2. max():**

It returns highest value in a set.

**3. min():**

It returns lowest value in a set.

**4. sum():**

It returns sum of set elements.

ex-1:

s = {78,34,567,12,3,89}

print('No. of elements = ',len(s))

print('Highest value = ',max(s))

print('Lowest value = ',min(s))

print('sum of set elements = ',sum(s))

**Inserting elements into a set:**

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

**5. add():**

==> This function is used to add only one element.

ex-1:

s = {11,22,33,44}

print(s)

s.add(99)

s.add(8😎

s.add(77)

s.add(66)

s.add(55)

print(s)

**6. update():**

==> This function is used to add morethan one element.

ex-1: using list

s = {11,22,33,44}

print(s)

s.update([99,88,77,66,55])

print(s)

ex-2: using tuple

s = {11,22,33,44}

print(s)

s.update((99,88,77,66,55))

print(s)

ex-3: using set

s = {11,22,33,44}

print(s)

s.update({99,88,77,66,55})

print(s)

ex-4: using range()

s = {11,22,33,44}

print(s)

s.update(range(1,11,1))

print(s)

ex-5: using string ()

s = {11,22,33,44}

print(s)

s.update('tronix')

print(s)

**7. pop():**

==> This function is used to remove random element from a set.

**8. remove():**

==> This function is used to remove given element from a set.

==> if the given element not present in a set, then,

it returns Error.

**9. discard():**

==> This function is used to remove given element from a set.

==> if the given element not present in a set, then, it returns None.

**10. clear():**

==> This function is used to remove all elements from a set.

Q.what are iterables in python:

* Iterable means group of values.
* python iterables are :

list

tuple

set

dictionary

range

string

'''

**11. copy():**

==> This function is used for copy the elements from

one set into antother set.

ex-1:

s1 = {11,22,33,44,55}

s2 = s1.copy()

print(s1)

print(s2)

**mathematical related functions in sets:**

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

1. set union()

2. set intersection()

3. set difference()

4. set symmetric\_difference()

5. issubset()

6. issuperset()

7. isdisjoint()

**1. set union() or |** **:**

==> This function, returns all elements present in both sets.

ex-1:

a = {11,22,33,44,55}

b = {77,55,66,99,11}

c = a.union(b)

print(c)

d = a|b

print(d)

**2. intersection() or & :**

==> This function, returns common elements in both sets.

ex-1:

a = {11,22,33,44,55}

b = {77,55,66,99,11}

c = a.intersection(b)

print(c)

d = a&b

print(d)

**3. difference() or - :**

==> This function, returns,

the elements present in first set but not in second set.

ex-1:

a = {11,22,33,44,55}

b = {77,55,66,99,11}

c = a.difference(b)

print(c)

d = a-b

print(d)

ex-2:

a = {11,22,33,44,55}

b = {77,55,66,99,11}

c = b.difference(a)

print(c)

d = b-a

print(d)

**4. symmetric\_difference() or ^ :**

==> This function, returns,

except common elements

ex-1:

a = {11,22,33,44,55}

b = {77,55,66,99,11}

c = a.symmetric\_difference(b)

print(c)

d = a^b

print(d)

write a program to find the given element is present or not?

method-1:

a=[11,22,33,44,55]

key = int(input('which element to be search ...'))

if key in a:

print('Given element is present')

else:

print('Given element is not present')

method-2:

a=[11,22,33,44,55]

found=0

key = int(input('which element to be search ...'))

for i in a:

if i==key:

found=1

break

if found:

print('Given element is present')

else:

print('Given element is not present')

**Q.**write a program to find common elements in 2 lists

method-1: Two lists are converting into sets and find the

intersection.

a = [11,22,33,44,55,66]

b = [33,99,88,11,100,200,300]

a = set(a)

b = set(b)

c=a.intersection(b)

print('common elements = ',c)

method-2:

a = [11,22,33,44,55,66]

b = [33,99,88,11,100,200,300]

c = [ ]

for i in a:

if i in b:

c.append(i)

print('common elements = ',c)

**Q.**write a program to find the difference between

first and last element in the given list

method-1:

a = [11,22,33,44,55,66]

d = a[0]-a[-1]

print('difference = ' ,d)

method-2:

a = [11,22,33,44,55,66,77,88,99]

d = a[0]-a[len(a)-1]

print('difference = ' ,d)

**Q.**write a program to print the duplicate elements

a = [11,22,33,44,22,22,11,1188,88,88,88,22,22,22,22]

duplist = [ ]

for i in a:

if a.count(i)>1:

duplist.append(i)

print('The Duplicate elements are : ',set(duplist))

**Q.**write a program to remove the duplicate elements.

a = [11,22,33,44,22,22,11,11,11,22,22,22,22]

print('Before removing duplicate elements the list = ',a)

a = set(a)

print('After removing duplicate elements the list = ',list(a))

write a program to print all index numbers of a given element

0 1 2 3 4 5 6 7 8 9 10

method-1: using for loop

a = [11,22,33,11,11,44,55,11,33,33,33]

ele = int(input('Enter Given element '))

print('Given element index numbers are: ')

for i in range(0,len(a),1):

if a[i]==ele:

print(i)

method-2: using while loop

a = [11,22,33,11,11,44,55,11,33,33,33]

ele = int(input('Enter Given element '))

print('Given element index numbers are: ')

i=0

while i<len(a):

if a[i]==ele:

print(i)

i=i+1

**Q.**write a program to find the smallest and largest numbers

a = [45,8,23,5,99,12]

a.sort()

print('Smallest value = ',a[0])

print('Largest value = ',a[-1])

**Q.**write a program to enter list elements at runtime using eval() function

a = eval(input('Enter List elements'))

print(a)

print(type(a))

**Q.**write a program to enter tuple elements at runtime using eval() function

a = eval(input('Enter tuple elements'))

print(a)

print(type(a))

**Q.**write a program to enter set elements at runtime using eval() function

a = eval(input('Enter set elements'))

print(a)

print(type(a))

a = eval(input('Enter set elements'))

print(a)

print(type(a))

**Q.**write a program to enter list elements at runtime

a = [ ]

n = int(input('How many elements are stored into a list'))

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

ele = int(input('Enter List element'))

a.append(ele)

print('Given List = ',a)

**Q.**write a program to remove a specific digit

from every element of the list. '''

**DICTIONARY DATA STRUCTURE**

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

properties(characterstics):

1. A group of key-value pairs, each pair separated by commas

and enclosed by curly braces. It is called dictionary.

key value

--- ------

htno --> student name

aadhar number --> citizen name

Flat No. --> resident name

account no. --> customer name

phone no. --> customer name

2. dictionariess are created by using dict()

3. All elements are stored orderly.

4. Insertions are allowed

5. Deletions are alllowed

6. Duplicate keys are not allowed.

but duplicate values are allowed.

7. dict does not supports indexing and slicing.

It supports only keys.

8. dict is mutable

9. dict is dynamic.

Based on our requirement the size can be either

increased or decreased.

10. dict supports heterogeneous elements

11. dict declaration syntax is:

dictname = { key-1:value-1,key-2:value-2, ....key-n:value-n }

ex-1: To create a empty dictionary using curly braces

d = { }

print(d)

print(type(d))

ex-2: To create a empty dictionary using dict()

d = dict()

print(d)

print(type(d))

ex-3: To create a student dictionary with keys & values

d = {1901:'ram',1902:'venkat',1903:'balu',1904:'sita' }

print(d)

print(type(d))

ex-4: To create a course dictionary with keys & values

d = {101:'python',102:'c',103:'c++',104:'java' }

print(d)

print(type(d))

ex-5: To create a dictionary with homogeneous keys

d = {101:'python',102:'c',103:'c++',104:'java' }

print(d)

print(type(d))

ex-6: To create a dict with heterogeneous keys

d = { 101:'python','c':'c','cpp':'c++',104:'java',10.5:'Django' }

print(d)

print(type(d))

ex-7: To create a dictionary with duplicate keys '

d = {101:'python',102:'c',103:'c++',104:'java',102:'ML',103:'AI'}

print(d)

print(type(d))

**Built-in functions or methods in dicts:**

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

1. len()

2. max()

3. min()

4. sum()

5. setdefault()

6. update()

7. pop()

8. popitem()

9. clear()

10. copy()

11. get()

12. keys()

13. values()

14. items()

15. fromkeys()

1. len():

It returns No. of elements in a dictionary.

2. max():

It returns highest key in a dictionary.

3. min():

It returns lowest key in a dictionary.

4. sum():

It returns sum of keys in a dictionary.

ex-1:

d = {101:'python',102:'c',103:'c++',104:'java' }

print(len(d))

print(max(d))

print(min(d))

print(sum(d))

**Inserting new elements into a dictionary:**

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

5. setdefault():

==> This function is used for,

inserting new key-value pair into a dictionary.

==> It is used to add only one pair.

==> To add new pair end of the dictionary.

==> syntax:

setdefault(key,value)

==> key is compulsory

value is optional.

==> if the value is not present, then,

by default, 'None' value set to the given key.

==> if the key is already present, then,

we cannot add new value to the existing key.

ex-1:

d = {101:'python',102:'c',103:'c++',104:'java'}

print(d)

d.setdefault(109,'AI')

d.setdefault(108,'ML')

d.setdefault(107,'DATA SCIENCE')

d.setdefault(106)

d.setdefault(105)

print(d)

ex-2:

d = {101:'python',102:'c',103:'c++',104:'java'}

print(d)

d.setdefault(102,'AI')

print(d)

6. update():

==> This function is used for,

inserting new key-value pairs into a dictionary.

==> It is used to add morethan one pair.

ex-1:

d = {101:'python',102:'c',103:'c++',104:'java'}

print(d)

d.update({109:'AI',108:'ML',107:'DATA SCIENCE',106:None,107:None} )

print(d)

ex-2:

d = {101:'python',102:'c',103:'c++',104:'java'}

print(d)

d1 = {109:'AI',108:'ML',107:'DATA SCIENCE',106:None,107:None}

d.update(d1)

print(d)

ex-3:

d = {101:'python',102:'c',103:'c++',104:'java'}

print(d)

d.update({102:'AI'})

print(d)

**Removing elements from a dictionary**

7. pop():

==> This function is used for remove the given key from a dict.

==> if the given key is present, then, remove it.

==> if the given key is not present, then,

it returns KeyError.

ex-1:

d = {101:'python',102:'c',103:'c++',104:'java'}

print(d)

d.pop(102)

d.pop(103)

d.pop(104)

print(d)

ex-2:

d = {101:'python',102:'c',103:'c++',104:'java'}

print(d)

d.pop(402)

d.pop(103)

d.pop(104)

print(d)

8. popitem() :

==> This function is used for remove the last element from a dict.

ex-1:

d = {101:'python',102:'c',103:'c++',104:'java'}

print(d)

d.popitem()

d.popitem()

d.popitem()

d.popitem()

d.popitem()

print(d)

9. clear():

==> This function is used to remove all elements from a dictionary.

ex-1:

d = {101:'python',102:'c',103:'c++',104:'java'}

print(d)

d.clear()

print(d)

10. copy():

==> This function is used to copy the elements

from a one dictionary into another dictionary

ex-1:

d = {101:'python',102:'c',103:'c++',104:'java'}

d1 = d.copy()

print(d)

print(d1)

11. get():

== >This function returns the value of the given key.

==> if the key is not present,

then, it returns, None.

ex-1:

d = {101:'python',102:'c',103:'c++',104:'java'}

print(d.get(102))

print(d.get(104))

print(d.get(101))

print(d.get(103) )

print(d.get(109))

12. keys():

==> It returns only keys.

ex-1:

d = {101:'python',102:'c',103:'c++',104:'java'}

#print(d.keys())

for i in d.keys(): # [101,102,103,104]

print(i)

13. values():

==> It returns only values.

==> It returns only keys.

ex-1:

d = {101:'python',102:'c',103:'c++',104:'java'}

#print(d.values())

for i in d.values(): # ['python','c','cpp','java]

print(i)

14. items():

==> It returns both.

i.e. keys & values

x-1:

d = {101:'python',102:'c',103:'c++',104:'java'}

#print(d.items())

#for i in d.items(): #(101, 'python'), (102, 'c'), (103, 'c++'), (104, 'java')])

# print(i)

for i,j in d.items(): #(101, 'python'), (102, 'c'), (103, 'c++'), (104, 'java')])

print(i,'------->',j)

15. fromkeys():

==> syntax:

dict.fromkeys(iterable,value)

Here,

iterable means either list/tuple/set/dict/range/string.

the value is optional. by default value is None.

ex-1:

a = [11,22,33,44,55]

d = dict.fromkeys(a)

print(d)

ex-2 To pay scholarship Rs. 2500 for the selected hall ticket numbers

htno = [191,2562,34343,4344,5434345]

d = dict.fromkeys(htno,2500)

print(d)

**what are iterables in python:**

The iterables returns group of values.

the iterables are:

list

tuple

set

dict

string

range

'''

d = {101:'python',102:'c',103:'c++',104:'java'}

x = d.keys()

y = d.values()

print(x)

print👍

'''

'''

ex: duplicates keys are not allowed.

d={101:'siva',101:'ram',101:'prasad'}

print(d)

ex: duplicates valuess are not allowed.

d={101:'siva',102:'siva',103:'siva'}

print(d)

'''

1. write a program to create a dictionary and

to enter dictionary elements at runtime

d = { }

x = int(input('Enter key'))

y = input('Enter value')

d[x] = y

print(d)

2. write a program to create a dictionary and

to enter dictionary elements at runtime, 'n' elements

d = { }

n = int(input('How many elements stored into a dictionary'))

for i in range(n):

x = int(input('Enter key'))

y = input('Enter value')

d[x] = y

print(d)

**3.** write a program to create a dictionary values using lists

d = {1901:['ram',77,88,99],1902:['sita',55,44,66]}

print(d.get(1901))

print(d.get(1902))

**STRINGS**

**or**

**string handling**

**or**

**string data type**

==> A single character or group of characters is called a string.

==> It allows alphabets, digits & special symbols.

==> The strings are enclosed by either

single quotes or double quotes or

triple single quotes or triple double quotes

ex: 'r'

'ram'

'python 3.11.0'

"Tronix technologies"

'''Tronix technologies'''

"""Tronix technologies"""

'python" --> invalid statements

"python' --> invalid statements

'''python""" --> invalid statements

**arithmetic operators in strings:**

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

**1.** string concatenation operator(+)

**2.** string repetition operator(\*)

1. string concatenation operator(+):

==> This operator is used to concatenate(join/merge)

2 or more strings.

ex-1:

s1 = 'python'

s2 = 'tronix'

s3 =s1+s2

print(s3)

ex-2:

s1 ='python'

s2 = 'tronix'

s3 =s1+' '+s2

print(s3)

ex-3:

s1 ='python'

s2 = 'tronix'

s3 =s1+'\t\t'+s2

print(s3)

2. string repetition operator(\*):

==> This operator is used to repeat a string 'n' number of times

ex-1:

s1 = 'python'

s2 = 'tronix'

s3 =s1+s2

print(s3\*5)

print(s1\*4)

print(s2\*10)

**relational operators in strings:**

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

<

>

==

!=

<=

>=

==> These operartors are used for comparison purpose.

ex-1:

s1='java'

s2='python'

s3='python'

print(s1<s2)

print(s1==s2)

print(s2==s3)

ex-1:

s1='java'

s2='cpython'

print(s1<s2)

print('level' > 'abcd')

print('java' > 'jpython')

print('aravind'<'arjun')

**membership operators in strings:**

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

==> These operators are used to find the string is present or not.

in --> It returns either True/False.

if the given value is present, then,

it returns True. otherwise it returns False.

not in --> It returns either True/False.

if the given value is not present, then,

it returns True. otherwise it returns False.

ex-1:

s = 'tronix'

print('r' in s)

print('R' in s)

print('x' in s)

print('N' in s)

print('o' not in s)

**To access string elements using indexing:**

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

===> A string is divided into smaller parts.

each part represents index number.

==> In strings, index numbers are 2:

positive indexing

negative indexing

positive indexing starts from left to right

i.e. 0,1,2,3,4.....size-1.

negative indexing starts from right to left

i.e. -1,-2,-3,....-size.

ex-1: To dispaly string elements using positive indexing

s = 'tronix'

print(s[0])

print(s[1])

print(s[2])

print(s[3])

print(s[4])

print(s[5])

print(s[6])

ex-2: To dispaly string elements using positive indexing

s = 'tronix'

print(s[-1])

print(s[-2])

print(s[-3])

print(s[-4])

print(s[-5])

print(s[-6])

**To access string elements using slicing:**

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

==> using indexing to access only one character.

==> if we want to access, morethan one character,

we go for slicing concept.

**To access string elements using slicing:**

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

==> using indexing to access only one character.

==> if we want to access, morethan one character,

we go for slicing concept.

==> syntax:

[strat:stop:step]

Here,

==> start indicates starting index number.

It is optional

By default, starting index number is '0'.

==> stop indicates ending index number.

It is optional

By default, end of the string.

==> step indicates, the index number either increment/decrement.

It is optional

By default, increased by '1'.

ex-1 :

s = 'python is a high level programming Language'

#print(s[0:9:1])

#print(s[0: : 1])

#print(s[0: : 4])

#print(s[ : : ])

#print(s[-1 : : -1 ])

print(s[ : : -1 ])

**built-in functions or methods in strings:**

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

1. len()

2. upper()

3. lower()

4. title()

5. capitalize()

6. swapcase()

7. isupper()

8. islower()

9. isdigit()

10. isspace()

11. isalpha()

12. isalnum()

13. istitle()

14. split()

15. join()

16. trim()

17. ltrim()

18. rtrim()

19. replace()

20. format()

21. count()

22. index()

1. len():

==> This function returns No. of characters in a string

ex-1:

s1 = 'python'

s2 = 'python tronix hyderabad'

print('No. of characters in string-1 = ', len(s1))

print('No. of characters in stirng-2 =',len(s2))

2. upper():

==> This function is used to convert a given string into upper case

ex-1:

s1 = 'python'

s2 = 'python tronix hyderabad'

print(s1.upper())

print(s2.upper())

3. lower():

==> This function is used to convert a given string into lower case

ex-1:

s1 = 'KUSU SRINIVAS'

print(s1.lower())

4. title():

==> This function is used to convert,

each and every word first letter into upper case

ex-1:

s1 = 'tronix is best'

s2 = 'tronix hyderabad'

print(s1.title())

print(s2.title())

5. capitalize():

==> This function is used to convert,

in the given string first letter convert into upper case

remaining characters into lower case.

ex-1:

s1 = 'tronix is best HYDERABAD'

s2 = 'tronix hyderabad'

print(s1.capitalize())

print(s2.capitalize())

6. swapcase() :

==> This function is used to convert,

upper case into lower case and

lower case into upper case

ex-1:

s1 = 'tRoNix'

print(s1.swapcase())

7. isupper():

It returns either True/False.

IN the given string, all characters are upper case, then,

it returns True. otherwise it returns False.

ex-1:

s1 = 'TRONIX'

s2 = 'tronix IS BEST'

s3 = 'hyderabad'

print(s1.isupper())

print(s2.isupper())

print(s3.isupper())

8. islower():

==> It returns either True/False.

In the given string, all characters are lower case, then,

it returns True. otherwise it returns False.

ex-1:

s1 = 'TRONIX'

s2 = 'tronix IS BEST'

s3 = 'hyderabad'

print(s1.islower())

print(s2.islower())

print(s3.islower())

9. isdigit():

==> It returns either True/False.

In the given string, all characters are digits, then,

it returns True. otherwise it returns False.

ex-1:

s1 = 'TRONIX'

s2 = 'tronix IS BEST'

s3 = 'hyderabad'

s4 = 'python 3.11.0'

s5 = '4567'

print(s1.isdigit())

print(s2.isdigit())

print(s3.isdigit())

print(s4.isdigit())

print(s5.isdigit())

ex-2 : write a program to find your phone number is valid or not

phno = input('Enter phone number')

if phno.isdigit() and len(phno)==10:

print('Valid phone number')

else:

print('Invalid phone number')

10. isspace():

==> It returns either True/False.

In the given string, all characters are white spaces(blank spaces),

then,

it returns True. otherwise it returns False.

ex-1:

s1 = 'TRONIX'

s2 = 'tronix IS BEST'

s3 = 'hyderabad'

s4 = 'python 3.11.0'

s5 = '4567'

s6 = ''

s7 = ' '

print(s1.isspace())

print(s2.isspace())

print(s3.isspace())

print(s4.isspace())

print(s5.isspace())

print(s6.isspace())

print(s7.isspace())

11. isalpha():

==> It returns either True/False.

In the given string, all characters are alphabets,then,

it returns True. otherwise it returns False.

ex-1:

s1 = 'TRONIX'

s2 = 'tronix IS BEST'

s3 = 'hyderabad'

s4 = 'python 3.11.0'

s5 = '4567'

s6 = ''

s7 = ' '

print(s1.isalpha())

print(s2.isalpha())

print(s3.isalpha())

print(s4.isalpha())

print(s5.isalpha())

print(s6.isalpha())

print(s7.isalpha())

12. isalnum() :

==> alnum means alphabets , numerics

==> It returns either True/False.

In the given string, all characters are either alphabets or digit,

then,

it returns True. otherwise it returns False.

ex-1:

s1 = 'TRONIX'

s2 = 'tronix IS BEST'

s3 = 'hyderabad'

s4 = 'python 3.11.0'

s5 = '4567'

s6 = ''

s7 = ' '

s8 = 'python310'

print(s1.isalnum())

print(s2.isalnum())

print(s3.isalnum())

print(s4.isalnum())

print(s5.isalnum())

print(s6.isalnum())

print(s7.isalnum())

print(s8.isalnum())

13. istitle():

==> It returns either True/False.

In the given string,

each and every word first character is upper case and

remaining characters in lower case, then,

it returns True. otherwise it returns False.

ex-1:

s1 = 'TRONIX'

s2 = 'tronix IS BEST'

s3 = 'hyderabad'

s4 = 'Tronix Is Best'

print(s1.istitle())

print(s2.istitle())

print(s3.istitle())

print(s4.istitle())

14. split():

==> this function is used to split the given string

using separator.

==> syntax:

split('separator')

==> By default , separator is blank space

ex-1:

s = 'python is high level programming langauge'

x = s.split()

print(x)

ex-2:

s = 'python is high level programming langauge'

x = s.split('h')

print(x)

ex-3:

s = 'python is high level programming langauge'

x = s.split('e')

print(x)

ex-4:

s = 'python is high level programming langauge'

x = s.split('i')

print(x)

ex-5:

s = 'python is high level programming langauge'

x = s.split('z')

print(x)

15. join()

16. strip()

17. lstrip()

18. rstrip()

19. replace()

20. format()

21. count()

22. index()

15. join():

==> To combine 2 or more strings

==> syntax:

'separator'.join('given string')

==> By default separator is blank space.

ex-1:

a = ['ram','siva','venkat','laxman']

s = ' '.join(a)

print(s)

ex-2:

a = ['ram','siva','venkat','laxman']

s = '-'.join(a)

print(s)

ex-3:

a = ['ram','siva','venkat','laxman']

s = ':'.join(a)

print(s)

ex-4:

d = ['21','11','2022']

s1 = '/'.join(d)

print(s1)

s2 = '-'.join(d)

print(s2)

16. strip():

This function is used to remove both side blank spaces.

i.e. leftside and right side

ex-1:

s = ' tronix '

print(len(s))

print(len(s.strip()))

17. lstrip():

This function is used to remove left side blank spaces.

ex-1:

s = ' tronix'

print(len(s))

print(len(s.lstrip()))

18. rstrip():

This function is used to remove right side blank spaces.

ex-1:

s = 'tronix '

print(len(s))

print(len(s.rstrip()))

19. replace():

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

==> This function is used for replace old string into new string.

==> sytnax:

replace('old string','new stirng',n)

Here,

n indicates number of times.

ex-1:

s = 'python'

print(s.replace('p','cp'))

print(s.replace('p','jp'))

ex-2:

s = 'rama is a good boy'

print(s.replace('rama','venkat'))

ex-3:

s = 'python is easy to learn'

print(s.replace('easy','difficult'))

ex-4:

s = '''java is a high level programming langauge

java is a oop

java is a pop

java is a dynamice proramming langauge

java is used to develop the web apps'''

print(s.replace('java','python'))

ex-5:

s = '''java is a high level programming langauge

java is a oop

java is a pop

java is a dynamice proramming langauge

java is used to develop the web apps'''

print(s.replace('java','python',3))

format():

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

ex-1:

a=10

b=20

c=45

print(a,b,c)

print('A value = ',a,'B value = ',b,'C value = ',c)

print('A value = {} B value = {} C value = {}'.format(a,b,c))

ex-2:

empid=101

ename='ram'

sal=99000

print('Employee Id = {},Employee Name = {},salary ={}'.format(empid,ename,sal))

ex-3:

eggs = 12

cost = 6

print('{} Eggs Total cost = {}'.format(eggs,eggs\*cost))

Here,

{} is called replacement operator

count():

==> This function is used to find ,

how many times a given character present in the string.

==> if the given character is not present in a string, then,

it returns zero.

ex-1:

s = 'python is a high level programming language'

print(s.count('h'))

print(s.count('i'))

print(s.count('y'))

print(s.count('t'))

print(s.count('z'))

index():

==> This function returns, index number of the given character .

==> if the given character is not present in a string, then,

it returns Error.

ex-1:

s = 'python is a high level programming language'

print(s.index('h'))

print(s.index('i'))

print(s.index('y'))

print(s.index('t'))

print(s.index('z'))

**Functions**

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

==> A group of statements are together into a single unit.

That unit is called a function.

==> python supports 2 types of functions:

1. built-in functions

2. user-defined functions

**1. Built-in functions:**

==> The functions which are coming from along with python software.

These functions are called built-in functions.

==> the built-in functions are also called

readymade functions/predefined functions/standard functions.

ex: print(),input(), type(),id(),

int(),float(),str(),complex(),bool(),

list(),tuple(),set(),dict(),

len(),max(),min(),sum(),

append(),extend(),insert(),

pop(),remove(),clear(),copy()

sort(),reverse(),count(),index()

popitem(),discard(),add(),update(),

upper(),lower(),isupper(),islower(),

isalpha(),isalnum(),istitle(),isdigit(),

format(),split(),join(),replace(),strip(),lstrip(),rstrip(),

capitalize(),swapcase(),.....etc2. user-defined functions:

**2. user-defined functions:**

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

==> The functions which are developed by programmer(developer).

These functions are called user-defined functions.

==> syntax:

def functionname(argumentslist):

st-1

st-2

st-3

---

---

st-n

return expression

Here,

==> def is a keyword.

It is indicates function definition/function declaration

==> functionname is the name of the function

==> argumentslist is a list of arguments.

i.e. input to the function

==> ':' indicates beging of the function

==> st-1,st-2,st-3,...st-n are called function body.

==> return expression is used to return

one value/morethan from function to main program.

ex-1: write a function to find square of a given number

def square(x):

y = x\*x

return y

n = int(input('Enter any number'))

s = square👎

print('Square value = ',s)

ex-2: write a function to find cube of a given number

def cube(x):

y = x\*x\*x

return y

n = int(input('Enter any number'))

c = cube👎

print('Cube value = ',c)

ex-3: write a function to find the addition of two numbers

def addition(x,y):

z = x+y

return z

a = int(input('Enter first number'))

b = int(input('Enter second number'))

c = addition(a,b)

print('Addition value = ',c)

1. write a function to find the area of a rectangle.

def rectangle(x,y):

z = x\*y

return z

l = int(input('Enter length value'))

b = int(input('Enter breadth value'))

area = rectangle(l,b)

print('Area of a rectangle = ',area)

2. write a function to find the area of a triangle.

def triangle(x,y):

z = x\*y/2

return z

b = int(input('Enter base value'))

h = int(input('Enter height value'))

area = triangle(b,h)

print('Area of a triangle = ',area)

3. write a function to find the area of a circle.

import math

def circle(x):

z = math.pi \* x\*\*2

return z

r = int(input('Enter radius value'))

area = circle(r)

print('Area of a circle = ',area)

4. write empty function

def wish():

pass

wish()

Note:

pass is a keyword .

This keyword is used to create empty function

5. write a function to print messages

def wish():

print('Good Morning To All')

print('Welcome To Tronix Technologies')

print('Welcome To python course')

wish()

**6.** write multiple functions to print different messages

def mng():

print('Good Morning To All')

def aft():

print('Good Afternoon To All')

def evg():

print('Good Evening To All')

def ngt():

print('Good Night To All')

mng()

aft()

evg()

ngt()

**7.** write multiple functions to find arithmetic operations

def add(x,y):

print(x+y)

def sub(x,y):

print(x-y)

def mul(x,y):

print(x\*y)

def div(x,y):

print(x/y)

add(11,22)

sub(6,3)

mul(4,7)

div(10,3)

Ex: To import the module

import tronix10am

tronix10am.mul(8,4)

32

tronix10am.add(4,6)

10

tronix10am.sub(5,2)

3

tronix10am.div(55,3)

18.333333333333332

tronix10am.mod(4,7)

AttributeError: module 'tronix10am' has no attribute 'mod'

**Types of user-defined functions:**

1. Functions with arguments with return values.

2. Functions with arguments no return values.

3. Functions no arguments with return values.

4. Functions no arguments with no values.

**return statement:**

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

==> return is a keyword.

==> This keyword is used,

to return values from function to main program.

==> It returns one value or morethan one value.

==> This keyword is used for inside a function.

==> syntax:

return expression

ex-1: To return only one value

def multiplication(x,y):

z=x\*y

return z

a = int(input('Enter first number'))

b = int(input('Enter second number'))

k = multiplication(a,b)

print('Multiplication value = ', k)

ex-2: To return morethan one value

method-1:

def calculation(x,y):

m = x+y

n = x-y

p = x\*y

q = x/y

return m,n,p,q

a = int(input('Enter first number'))

b = int(input('Enter second number'))

c,d,e,f = calculation(a,b)

print('Addition value = ',c)

print('subtraction value = ',d)

print('Multiplication value = ', e)

print('division value = ',f)

method-2:

def calculation(x,y):

return x+y,x-y,x\*y,x/y

a = int(input('Enter first number'))

b = int(input('Enter second number'))

c,d,e,f = calculation(a,b)

print('Addition value = ',c)

print('subtraction value = ',d)

print('Multiplication value = ', e)

print('division value = ',f)

method-3:

def calculation(x,y):

m = x+y

n = x-y

p = x\*y

q = x/y

return m,n,p,q

a = int(input('Enter first number'))

b = int(input('Enter second number'))

c = calculation(a,b)

print(c)

print(type(c))

ex-3: write a program to return morethan one value using strings

def find(x):

return len(x),x.upper(),x.title(),

a,b,c = find('tronix')

print(a)

print(b)

print(c)

ex-4: write a program to return morethan one value using lists

def find(x):

return len(x),max(x),min(x),sum(x)

a,b,c,d = find([3,78,2,4,9])

print(a)

print(b)

print(c)

print(d)

**Q.**write a program to find the biggest of two numbers

def biggest(x,y):

if x>y:

print('First number is biggest')

elif y>x:

print('Second Number is biggest')

elif x==y:

print('Both are equal')

biggest(45,3)

biggest(67,99)

biggest(12,12)

**Q.**write a program to find the biggest of two numbers

def biggest(x,y):

if x>y:

return 'First number is biggest'

elif y>x:

return 'Second Number is biggest'

elif x==y:

return 'Both are equal'

print(biggest(45,3))

print(biggest(67,99))

print(biggest(12,12))

**comprehensions**

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

==> To store the values, simple and easy way.

==> python supports:

1. list comprehension

2. tuple comprehension

3. set comprehension

4. dict comprehension

1. list comprehension:

==> To create a list simple and easy way.

==> syntax:

listname = [ expression for loop if condition ]

ex-1: write a program to create a list and store 1 to 10 numbers.

method-1:

a = [1,2,3,4,5,6,7,8,9,10]

print(a)

method-2: using list() & range()

a = list(range(1,11,1))

print(a)

method-3: using for loop

a = [ ]

for i in range(1,11,1):

a.append(i)

print('Orignal List = ',a)

method-4: using list comprehension

a = [ i for i in range(1,11,1) ]

print(a)

print(type(a))

2. write a to create list, squares of numbers

a = [ i\*i for i in range(1,11,1) ]

print(a)

3. write a to create list, cubes of numbers

a = [ i\*i\*i for i in range(1,11,1) ]

print(a)

4. write a program to store in between 1 to 10 , even numbers stored into list

a = [ i for i in range(1,11,1) if i%2==0]

print(a)

5. write a program to store in between 1 to 10 , odd numbers stored into

list

a = [ i for i in range(1,11,1) if i%2!=0]

print(a)

6. write a program to print the names whose names start with 's'

method-1: using list comprehension

names = ['srinivas','ram kumar','sridevi','venkat','sirisha' ]

a = [ i for i in names if i[0]=='s' ]

print(a)

method-2: without list comprehension

a = [ ]

names = ['srinivas','ram kumar','sridevi','venkat','sirisha' ]

for i in names:

if i[0]=='s':

a.append(i)

print(a)

2. tuple comprehension:

==> To create a tuple simple and easy way.

==> syntax:

tuplename = ( expression for loop if condition )

ex-1: write a program to create a tuple and store 1 to 10 numbers.

a = ( i for i in range(1,11,1) )

print(a)

print(type(a))

Note: python does not support tuple comprehension.

It supports generators.

3. set comprehension:

==> To create a set, simple and easy way.

==> syntax:

setname = { expression for loop if condition }

ex-1: write a program to create a tuple and store 1 to 10 numbers.

s = { i for i in range(1,11,1) }

print(s)

print(type(s))

4. dictionary comprehension:

==> To create a dictionary, simple and easy way.

==> syntax:

dictname = { expr-1:expr-2 for loop if condition }

Here,

expr-1 , it represents keys

expr-2 , it represetns values

ex-1:

d = { i:i for i in range(1,11,1) }

print(d)

ex-2:

d = { i:i\*i for i in range(1,11,1) }

print(d)

ex-3:

d = { i:i\*i\*i for i in range(1,11,1) }

print(d)

ex-4 : '

names = ['srinivas','ram kumar','sridevi','venkat','sirisha' ]

a = { i:i[0] for i in names }

print(a)

ex-5 :

names = ['srinivas','ram kumar','sridevi','venkat','sirisha' ]

a = { i:i[-1] for i in names }

print(a)

ex-6 :

names = ['srinivas','ram kumar','sridevi','venkat','sirisha' ]

a = { i:len(i) for i in names }

print(a)

**Types of arguments:**

**================================**

what is argument?

==> input to the function.

It is called argument.

==> python supports 2 Types of arguments:

1. actual arguments

2. formal arguments

1. actual arguments or actual parameters:

==> The variables which are used in function calling.

These arguments are called actual arguments

2. formal arguments or formal parameters

==> The variables which are declared in function definition.

These arguments are called formal arguments.

ex-1:

def multiplication(x,y): #function declaration function definition

z = x\*y

return z

a = int(input('Enter first number'))

b = int(input('Enter second number'))

c = multiplication(a,b) --> function call

print(c)

Here,

a,b are caleld actual arguments or actual parameters.

x,y are called formal arguments or formal parameters.

==> Again these actual and formal arguments

are divided into 5 types:

1. positional arguments

2. keyword arguments

3. default arguments

4. variable length arguments

5. keyword-variable length arguments

1. positional arguments:

a). The NO. of actual arguments and

the No. of formal arguments must be equal.

b). The position is also important.

ex-1:

def multiplication(x,y):

z = x\*y

return z

a = int(input('Enter first number'))

b = int(input('Enter second number'))

c = multiplication(a,b)

print(c)

ex-2:

def display(x,y,z):

print('Employee Number = ',x)

print('Employee Name = ',y)

print('Employee salary = ',z)

empid = 101

ename = 'ram'

sal = 67000

display(empid,ename,sal)

**2. keyword arguments:**

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

a). The NO. of actual arguments and

the No. of formal arguments must be equal.

b). The position is not important.

c). The value is assigned to the formal parameter.

ex-1:

def display(x,y,z):

print('Employee Number = ',x)

print('Employee Name = ',y)

print('Employee salary = ',z)

empid = 101

ename = 'ram'

sal = 67000

display(z=sal,x=empid,y=ename)

ex-2:

def calculate(x,y,z,m,n):

total = x+y+z

avg = total/3

print('student name = ',n)

print('Total marks = ',total)

print('average marks = ',avg)

htno = 190111

name = 'siva kumar'

m1=77

m2=88

m3=99

calculate(m=htno,n=name,x=m1,y=m2,z=m3)

3. Default arguments:

a). The number of arguments same or not, is not important

b). The position is also not important

c). sometimes we may want to use parameters in a function that takes default values in

case the user does not want to provide a value for them.

d). For this, we can use default arguments which assumes a default value if a value is not

supplied as an argument while calling the function.

e). An assignment operator '=' is used to give a default value to an argument.

f). In parameters list, we can give default values to one or more parameters.

Ex-1:

def display(x='Gudio Van Rossum',y=70):

print('Name of the person=',x)

print('Age of the person=',y)

display('ramkumar',40)

display('venkat',55)

display()

Ex-2:

def country(name='INDIA'):

print('current country name = ',name)

country( )

country('NEWYORK')

country('CHINA')

country()

Ex-3:

def printlist(upperlimit=4):

print('upper limit = ',upperlimit)

list1=list(range(upperlimit))

print('list = ',list1)

printlist ()

printlist(5)

printlist(3)

printlist( )

Ex-4: one argument is already deined in a formal argument,

and only one argument has to pass.

def person(name,age=45):

print(name)

print(age)

person(“kusu Srinivas”

4. Variable length arugments:

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

a). The number of arguments is not important

b). The position is also not important

c). Variable length arguments are also known as arbitrary arguments.

d). Sometimes, we do not know in advance the number of arguments that will be passed

into a function.

e). python allows us to handle this kind of situation through function calls with arbitrary

number of arguments or variable length argument.

f). In the function definition we use an asterisk (\*) before the parameter name to denote

this kind of argument.

g). This type of argument is used to pass multiple values.

Ex-1:

def calculate(\*x):

sum=0

print(x)

print(type(x))

for i in x:

sum=sum+i

print('sum of elements=',sum)

calculate(11,22,33,44)

calculate(5,8)

calculate(1,2,3,4,5,6,7,8,9,10)

calculate()

calculate(11)

calculate(5,8,9)

Ex-2:

def printnames(\*x):

for i in x:

print(i)

printnames('ram')

printnames('laxman','anji')

printnames()

printnames('ravana','bharat','laxman','anji')

Ex-3:

def largest(\*x):

return max(x)

print(largest(20,30))

print(largest(2,5,3))

print(largest(9,45,12,18,6))

print(largest(10,40,80,50))

print(largest(16,3,12,44,40))

By Mr. Kusu Srinivas

+91-8977525889 info@tronixtechs.com www.tronixtechs.com Page 104

Ex-4:

def sum(x,\*y):

a=x+y

print(a)

sum(5,6,7,2,8)

Here, the tip is that when we pass multiple values, then

the first value goes to the first formal argument, and

the star used with the second formal argument is to

accept all other values in it.

Demo for positional arguments

def display(x):

print(x)

display(25)

Demo for keyword arguments

def display(x):

print(x)

display(x=25)

Demo for default arguments

def display(x=3):

print(x)

display(25)

display()

display(10)

display(56)

display()

Demo for variable length arguments

def display(\*x):

print(x)

display(25)

display()

display(10,56,3232,123,23,213,2)

display(56,23)

display()

5. keyword-variable length arguments :

a). The No. of actual and forma arguments are different.

b). the position is also different.

c). We can pass any No. of keys and values.

This process keyword-variable length arguments.

d). The declaration of formal arguments is :

def functionname(\*\*argument):

body of the function

Here,

The formal argument is dictionary type.

ex-1:

def display(\*\*x):

print(x)

print(type(x))

display(a=25,b=10,c=9)

ex-2:

def f1(\*\*x):

print(x)

f1(a=12,b=16,c=89)

f1()

f1(m=656,n=54,p=4,q=3,r=43)

ex-2:

def f1(\*\*x):

for i,j in x.items():

print(i,'--------',j)

f1(a=12,b=16,c=89)

f1()

f1(m=656,n=54,p=4,q=3,r=43)

ex-4: To display student information

def displaystudentinfo(\*\*s):

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

for i,j in s.items():

print(i,'---------->',j)

displaystudentinfo(htno=101,name='siva',m1=77,m2=88,m3=99)

displaystudentinfo(htno=102,name='balu')

displaystudentinfo(htno=101,name='satya',m1=99,m2=99,m3=99)

displaystudentinfo(htno=105,name='mahesh',city='hyd')

write a program demo for variable length arguments .

def sum(\*args):

resultfinal = 0

for arg in args:

resultfinal = resultfinal + arg

return resultfinal

print(sum(10, 20))

print(sum(10, 20, 30))

print(sum(10, 20, 2))

write a program demo for variable length.

To find multiplication of all arguments

def multiplier(\*num):

prod = 1

for i in num:

prod = prod \* i

print("Product:",prod)

multiplier(3,5)

multiplier(1,2,4)

multiplier(2,2,6,7)

write a program demo for \*\*arguments

def myPrg(\*\*kwargs):

for k, v in kwargs.items():

print (k, "==" , v)

myPrg(first ='Hello', mid ='Welcome', last='Hello')

tuple packing and unpacking:

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

what is tuple packing ?

==> A group of values are stored into a

single variable. It is called packing.

ex-1:

a=11

b=22

c=33

d=44

e=55

t=a,b,c,d,e

print(t)

ex-2:

x='mangoes'

y='grapes'

z='banana'

m='orange'

fruits = x,y,z,m

print(fruits)

what is tuple unpacking ?:

==> A group of values are separated into

different items.

it is called tuple unpacking.

ex-1:

t=(10,20,30,40,50,60,70)

a,b,c,d,e,f,g = t

print(a)

print(b)

print(c)

print(d)

print(e)

print(f)

print(g)

'''

normal functions:

==> Normal functions are created by using 'def' keyword.

==> In normal functions, functionname is compulsory.

==> In normal functions,

no arguments/one argument/morethan one argument.

==> Normal functions allowed morethan one statement.

==> It allows return statement.

==> syntax:

def functionname(argumentslist):

st-1

st-2

st-3

--

--

st-n

return expression

lambda functions:

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

==> lambda functions are created by using 'lambda' keyword.

==> In lambda functions, functionname is not required.

i.e. lambda functions are also called

nameless functions or anonymous functions.

==> In lambda functions,

no arguments/one argument/morethan one argument.

==> lambda functions allowed only one statement.

==> Does not allow return statement.

==> syntax:

lambda argumentslist : expression

Here,

==> lambda is a keyword

==> argumentslisgt representes list of keywords.

==> ':' begining of the lambda function.

==> expresion represetns st-1

ex-1: write lambda function to find square of a given number

method-1: using lamda function

s = lambda x : x \* x

n = int(input('Enter any number'))

print('square value = ',s(n))

method-2: using normal function

def square(x):

y = x \* x

return y

n = int(input('Enter any number'))

s = square👎

print('square value = ',s)

ex-2: write lambda function to find cube of a given number

c = lambda x : x \* x \* x

n = int(input('Enter any number'))

print('square value = ',c(n))

ex-3: write lambda function to find addition of a two numbers

c = lambda x , y : x + y

a = int(input('Enter first number'))

b = int(input('Enter second number'))

print('Addition value = ',c(a,b))

ex-4: write lambda function to find area of a rectangle

area = lambda x , y : x \* y

l = int(input('Enter length value'))

b = int(input('Enter breadth value'))

print('Area of a rectangle = ',area(l,b))

ex-5: write lambda function to find biggest of two numbers

biggest=lambda x,y: x if x>y else y

a = int(input('Enter first number'))

b = int(input('Enter second number'))

print('Biggesgt value = ', biggest(a,b))

ex-6: write lambda function to find,

given number is even or odd number

findevenodd = lambda x: 'Even' if x%2==0 else 'odd'

n = int(input('Enter any number'))

print(findevenodd(n))

ex-7: write lambda function to find,

you are elegible for voting or not

voting = lambda x: 'Elegible' if x>=18 else 'Not Elegible'

age = int(input('Enter your age'))

print(voting(age))

Nested Functions:

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

==> A function within another function.

==> this process is called nested function.

==> syntax:

def outerfunctionname(argumentslist):

def innerfunctionname-1(argumentslist):

body of the function

def innerfunctionname-2(argumentslist):

body of the function

--

--

def innerfunctionname-n(argumentslist):

body of the function

programs:

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

1. write a program to find the reverse numver of a given number

n = int(input('Enter any number'))

revnum = 0

while n>0:

d = n%10

revnum = revnum\*10 + d

n = n//10

else:

print('Reverse number = ',revnum)

2. write a program to find the given number is palindrom .

Hint: if original number and reverse number, both are same,

then, the given number is palindrom number.

ex-1: original number ==> 345

reverse number ==> 543

It is not a palindrom number

ex-2: 1991 ==> 1991

It is a palindrom number

MADAM ==> MADAM

DAD ==> DAD

MOM ==>MOM

LEVEL ==> LEVEL

MALAYALAM ==> MALAYALAM

n = int(input('Enter any number'))

temp = n

revnum = 0

while n>0:

d = n%10

revnum = revnum\*10 + d

n = n//10

else:

print('Reverse number = ',revnum)

if revnum == temp:

print('Given number is palindrom number')

else:

print('Given number is not a palindrom number')

write a program ,swapping of two numbers

method-1: using temp variable

a = 9

b = 5

print('Before swapping ')

print('A value = ',a)

print('B value = ',b)

temp = a

a = b

b = temp

print('After swapping ')

print('A value = ',a)

print('B value = ',b)

method-2: without using temp variable

a = 10

b = 20

print('Before swapping ')

print('A value = ',a)

print('B value = ',b)

a = a+b

b = a-b

a = a-b

print('After swapping ')

print('A value = ',a)

print('B value = ',b)

method-3: only one line

a = 10

b = 20

print('Before swapping ')

print('A value = ',a)

print('B value = ',b)

a,b=b,a

print('After swapping ')

print('A value = ',a)

print('B value = ',b)

In javascript:

<script>

a = 10

b = 20

document.write("Before swapping "+"<br>");

document.write("A value = "+a+"<br>");

document.write("B value = "+b+"<br>");

[a,b]=[b,a]

document.write("After swapping "+"<br>");

document.write("A value = "+a+"<br>");

document.write("B value = "+b+"<br>");

</script>

Types of variables

or

variables scope

or

visibility of variables:

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

==> In Functions, the variables are divided into 3 types:

1. local variables

2. global variables

3. nonlocal variables

1. local variables:

==> The variables which are declared inside a function.

==> These variables are called local variables.

==> These variables are accessed inside a function.

==> These are not accessible outside of a given function.

ex-1: y):

z = x+y

return z

a = int(input('Enter first number'))

b = int(input('Enter second number'))

c = addition(a,b)

print(c)

Here,

a,b are called actual arguments

x,y are called formal arguments

z is called a local variable.

ex-2:

def calculate():

a=6

b=3

c=5

d=1

print(a+b+c+d)

calculate()

Here,

a,b,c,d variables are called local variables.

ex-3:

def f1():

print('I am begining f1 ...')

x=10

print(x)

print('I am End of the f1 ...')

def f2():

print('I am begining f2 ...')

y=20

print👍

print('I am End of the f2 ...')

def f3():

print('I am begining f3 ...')

z=25

print(z)

print('I am End of the f3 ...')

f1()

f2()

f3()

Here,

x,y,z are called local variables.

Because , these variables are declared inside a function.

2. global variables:

==> The variables which are declared outside a function.

==> These variables are called global variables.

==> These variables are used any where in the program.

i.e. inside the funciton or outside the funciton.

ex-1:

m=100

n=200

k=300

def find(x,y,z):

result = x+y+z+m+n+k

print(result)

find(11,22,33)

Here,

m,n,k are called global variables.

Becuase, these variables are declared outside of a function.

i.e. begining of the program.

Note:

x,y,z are called formal arguments

m,n,k are called global variables

result is called local variable.

ex-2:

k = 100

m = 200

n = 300

def f1():

print('I am begining f1 ...')

x=10

print(x)

print(m)

print👎

print(k)

print('I am End of the f1 ...')

def f2():

print('I am begining f2 ...')

y=20

print👍

print(m)

print👎

print(k)

print('I am End of the f2 ...')

def f3():

print('I am begining f3 ...')

z=25

print(z)

print(m)

print👎

print(k)

print('I am End of the f3 ...')

f1()

f2()

f3()

print('In main program ...')

print(m)

print👎

print(k)

Here,

x,y,z are called local variables

m,n,k are called global variables

ex: write a program demo for global variables.

k = 5

def display():

print(k)

def displayinfo():

print(k)

print(k)

display()

displayinfo()

print(k)

global statement:

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

==> global is a keyword.

==> This keyword is used for:

a). To declare global variables inside a function.

b). To modify the global variable value.

a). To declare global variables inside a function.

==> The global keyword is used to declare global variables

inside a function.

==> declaration syntax is :

global variablename

ex:

def display():

global k

k = 5

print(k)

def displayinfo():

print(k)

display()

displayinfo()

print(k)

b). To modify the global variable value:

def display():

global k

k = 5

print(k)

def displayinfo():

global k

k = k + 10

print(k)

display()

displayinfo()

print(k)

write a program to perform bank operations

accountcreate()

deposit()

withdraw()

checkbalance()

displayaccountinfo()

def accountcreate():

global accno

global cname

global bal

accno = 19012345

cname = 'prasa'

bal = 10000

print('Congrats , your account created ..')

def deposit():

global bal

amt = int(input('Enter deposit amount'))

bal = bal + amt

print('Your amount deposited successfully ...')

def withdraw():

global bal

amt = int(input('Enter withdraw amount'))

if amt<=bal:

bal = bal - amt

print('please collect your cash ...')

else:

print('Insufficient Funds')

def checkbalance():

print('Available balance = ',bal)

def displayaccountinfo():

print('Account Number = ',accno)

print('Customer Name = ',cname)

print('Available balance = ',bal)

while True:

print('1. To create account')

print('2. To deposit your amount')

print('3. To withdraw the amount')

print('4. Check balance')

print('5. To display Account Information')

print('6. Exit')

ch = int(input('Enter your choice'))

if ch==1:

accountcreate()

elif ch==2:

deposit()

elif ch==3:

withdraw()

elif ch==4:

checkbalance()

elif ch==5:

displayaccountinfo()

elif ch==6:

break

else:

print('Invalid operation ...')

ex: To modify global variable values inside a function

x = 9

y = 3

def calculate():

global x

global y

print(x\*y)

x = x + 25

y = y - 20

calculate()

print(x)

print👍

local variables --> These variables are declared inside a function

global variables --> These variables are declared outside a function.

--> The variables are declared by using global keyword.

nonlocal variable:

==> It is not a local variable.

==> It is not a global variable.

==> This variable belongs to outer function.

==> nonlocal variables are declared

within the nested functions only.

==> syntax:

nonlocal variablename

ex-1:

def f1():

k=9

def f2():

nonlocal k

print(k)

k=k+5

print(k)

f2()

f1()

ex-2:

m = 88

def calculate():

x = 10

print(x)

print(m)

def modify():

global m

nonlocal x

y = 20

print👍

print(x)

print(m)

x = x+60

m = m+45

modify()

print(x)

print(m)

calculate()

print(m)

Function Aliasing:

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

def display():

print('I am display ...')

display()

x = display

x()

hello = display

hello()

y = display

y()

Recursive functions:

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

==> A function calling itself.

This process is called a recursion.

ex-1:

def display():

print('I am display function ...')

display()

display()

The above function calling itself , 1010 times,

then, it returns RecursionError ''

ex-2: write a program to find sum of 1 to 10 numbers

method-1: using for loop

sum=0

for i in range(1,11,1):

sum = sum + i

print('sum of 1 to 10 numbers = ',sum)

method-2: using recursive function

def sumonetoten(x):

if x>10:

return 0

else:

return x + sumonetoten(x+1)

print('sum of 1 to 10 numbers = ',sumonetoten(1))

**Q.**write a recursion function to print 1 to 10 numbers

def sudheer(n):

sif n>=11:

return

else:

print( n)

return sudheer(n+1)

sudheer(1)

**Exception handling**

**or**

**Error handling**

what is an exception?

==> An Exception is an error that happens

during the execution of a program.

==> An exception is an event, which occurs during the

execution of a program,

that disrupts the normal flow of the program's instructions.

==> An exception is a Python object that represents an error.

==> When a Python script raises an exception,

it must either handle the exception immediately otherwise

it terminates and quits.

In python, there are 3 types of errors.

1. syntax error

2. logical error or sematic errors

3. runtime error

1. syntax error:

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

These are most basic type of errors.syntax errors are

almost always highlighted in IDLE, Where the syntax

error is. most syntax errors are typing mistakes,

incorrect indentation,incorrect arguments.

ex:

1. print('hello good morning)

2. print('hai friends welcome to hyd'

3. if a<b

print('a is smallest')

else

print('b is smallest')

4. i=1

while i<=10

print(i)

i=i+1

2. logical error:

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

your program might run without crashing(no syntax or

runtime errors), but still do the wrong output.

these type of errors are called logical errors.

ex-1:write a progam addition of two numbers.

a=10

b=20

c=a\*b

print(c)

3. runtime errors:

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

your program run without syntax errors and logical errors,

but while running the program it returns a errors.

These errors are called runtime errors.

the common runtime errors are:

NameError

TypeError

IndexError

ValueError

KeyError

AttributeError

ZeroDivisionError

SystemError

FileNotFoundError

FileExistsError

RecursionError

ex-1: Write a program demo for NameError

Note: If the given variable is not declared , then,

it returns NameError

while i<=10:

print(i)

i=i+1

ex-2: Write a program demo for TypeError

Note: If the both the variables are different data type,

it returns TypeError.

a = 'python'

b = 9.2

c = a + b

print(c)

ex-3: Write a program demo for IndexError

Note: If the given index number is not present in a list, then,

it returns IndexError

a = [11,22,33,44,55]

print(a[0])

print(a[2])

print(a[36])

ex-4: Write a program demo for ValueError

Note: If the given value is not present in a list, then,

it returns ValueError.

a = [11,22,33,44,55]

print(a)

a.remove(404)

print(a)

ex-5: Write a program demo for KeyError

Note: If the given value is not present in a set, then,

it returns KeyError.

a = { 11,22,33,44,55 }

print(a)

a.remove(404)

print(a)

ex-6: Write a program demo for KeyError

Note: If the given key is not present in a dictionary, then,

it returns KeyError.

d = {101:'python',102:'c',103:'java' }

print(d[101])

print(d[45])

ex-7: Write a program demo for AttributeError

Note: If the given method(function) is not present, then,

it returns AttributeError.

a = ( 11,22,33,44,55 )

print(a)

a.remove(404)

print(a)

ex-8: Write a program demo for ZeroDivisionError

Note: If the second operand is zero ,then,

it returns ZeroDivisionError.

a = 10

b = 0

c = a/b

print(c)

How to handle runtime errors:

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

==> The runtime errors can be handle by using

try-except-else -finally keywords.

==> syntax:

try:

Here we can develop

actual executable code.

except:

if any error raised in try block,

then automatically execute this block.

this block is compulsory block.

else:

if error not raised in try block,

then automatically execute this block.

This block is optional block.

finally:

if error raised or not raised in try block,

then automatically execute this block.

this block is optional block.

try

|

|

|

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

| |

| |

| |

except else

| |

| |

|---------finally-----------|

ex-1: Write a program demo for NameError using try and except blocks

Note: If the given variable is not declared , then,

it returns NameError

try:

#i=1

while i<=10:

print(i)

i=i+1

except NameError as x:

print('your program run with error')

print(x)

else:

print('your program success ....')

finally:

print('Thank u ')

ex-2: Write a program demo for TypeError using try,except blocks

Note: If the both the variables are different data type,

it returns TypeError.

try:

a = 'python'

#b = 'tronix'

b = 9.2

c = a + b

print(c)

except TypeError as x:

print('your program run with error')

print(x)

else:

print('your program success ....')

finally:

print('Thank u ')

ex-3: Write a program demo for IndexError using try,except blocks

Note: If the given index number is not present in a list, then,

it returns IndexError

try:

a = [11,22,33,44,55]

print(a[0])

print(a[2])

#print(a[36])

except IndexError as msg:

print('your program run with error')

print(msg)

else:

print('your program success ....')

finally:

print('Thank u ')

ex-4: Write a program demo for ValueError using try,except blocks

Note: If the given value is not present in a list, then,

it returns ValueError.

try:

a = [11,22,33,44,55]

print(a)

#a.remove(44)

a.remove(404)

print(a)

except ValueError as msg:

print('your program run with error')

print(msg)

else:

print('your program success ....')

finally:

print('Thank u ')

ex-5: Write a program demo for KeyError using try,except blocks

Note: If the given value is not present in a list, then,

it returns KeyError.

try:

a = { 11,22,33,44,55 }

print(a)

#a.remove(44)

a.remove(404)

print(a)

except KeyError as msg:

print('your program run with error')

print(msg,'not present in the given set')

else:

print('your program success ....')

finally:

print('Thank u ')

ex-8: Write a program demo for ZeroDivisionError

using try-except blck

Note: If the second operand is zero ,then,

it returns ZeroDivisionError.

try:

a = 10

#b = 5

b = 0

c = a/b

print(c)

except ZeroDivisionError as msg:

print('your program run with error')

print(msg)

else:

print('your program success ....')

finally:

print('Thank u ')

write a program demo for multiple exceptions

try:

a=9

c=a+b

print(c)

x=[11,22,33,44,55]

print(x[0])

print(x[5])

d=0

m = a/d

except NameError as msg:

print(msg)

except IndexError as msg:

print(msg)

except ZeroDivisionError as msg:

print(msg)

except TypeError as msg:

print(msg)

except ValueError as msg:

print(msg)

except KeyError as msg:

print(msg)

except AttributeError as msg:

print(msg)

except RecursionError as msg:

print(msg)

write a program demo for multiple exceptions,

only one except

try:

a=9

c=a+b

print(c)

x=[11,22,33,44,55]

print(x[0])

print(x[5])

d=0

m = a/d

except (NameError,IndexError,ZeroDivisionError,

TypeError,ValueError, KeyError, AttributeError) as msg :

print(msg)

write a program demo for multiple exceptions

try:

z = [11,22,33,44,55]

s = {11,22,33,44,55}

a=10

b=20

k=0

c='python'

d=a+b

print(d)

#e=b+c

#print(e)

print(k)

#print👍

#p=a/k

#print(p)

#print(z[10])

#z.remove(88)

s.remove(88)

except ZeroDivisionError as x:

print(x)

except TypeError as x:

print(x)

except IndexError as x:

print(x)

except ValueError as x:

print(x)

except KeyError as x:

print(x)

except NameError as x:

print(x)

write a program demo for multiple exceptions,

using single except statement .

try:

z = [11,22,33,44,55]

s = {11,22,33,44,55}

a=10

b=20

k=0

c='python'

d=a+b

print(d)

#e=b+c

#print(e)

print(k)

#print👍

p=a/k

print(p)

#print(z[10])

#z.remove(88)

#s.remove(88)

except (ZeroDivisionError,TypeError,IndexError,

ValueError,KeyError,NameError) as x:

print(x)

**Raising an Exception or To create user-defined exceptions:**

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

We can use raise keyword to throw an exception if a condition

occurs. The statement can be complemented with a custom

exception.

raise..............> Exception

If you want to throw an error when a certain condition

occurs using raise, you could go about it like this:

ex-1:

x = int(input('Enter x value'))

if x > 5:

raise Excepti) on('x value must be below 5')

else:

print('The value of x is =',x

ex-2:

n = int(input('Enter any number'))

if n<=0:

raise Exception('Does not allow negative numbers')

else:

print('Accepted....')

note: The raise keyword is used to raise an exception.

you can define what kind of error to raise and the text

to print to the user.

ex-2:

x = int(input('enter x value'))

if x < 0:

raise Exception('sorry, no numbers below zero')

else:

print('The value of x is =',x)

#raise keyword:

ex-3:

p=int(input('enter your pin number'))

if p==1234:

print('Login success...goto next step')

else:

raise Exception('wrong pin....')

ex-4:

accno=19011

cname='siva'

bal=25000

wamt=float(input('Enter withdrawal amount'))

if wamt > bal:

raise Exception('Insufficient Funds.....')

else:

bal=bal-wamt

print('pls collect cash.....')

print('After withdrawal present balance is:',bal)

ex-5:

a=10

#b=10

b=0

if b==0:

raise Exception('second operand is zero')

else:

c=a/b

print(c)

ex:

age = int(input('Enter your age '))

if age<=0:

raise Exception('Negative ages are not allowed ...')

elif age>0 and age<18:

print('you are not elegible for voting')

else:

print('you are elegible for voting ')sss

**Assertions in python**

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

==> The assert keyword is used when debugging code.

==> The assert keyword lets you test if a condition

in your code returns True,

if not, the program will raise an AssertionError.

what is an assertions?

==> In simpler terms, we can say that assertion

is the boolean expression

that checks if the statement is True or False.

If the statement is true then it does nothing and

continues the execution,

but if the statement is False then

it stops the execution of the program and throws an error.

==> You can write a message to be written if the code returns False,

check the example below.

ex-1:

x = "Hello"

#if condition returns False, AssertionError is raised:

assert x == "goodbye", "x should be 'hello'"

ex-2: Python assert keyword without error message

a = 4

# using assert to check for 0

print("The value of a / b is : ")

b = 2

assert b != 0

print(a / b)

ex-3: Python assert keyword with error message

a = 4

b = 0

print("The value of a / b is : ")

assert b != 0, "Zero Division Error"

print(a / b)

ex-4: Practical Application

This has a much greater utility in Testing and Quality Assurance role

in any development domain.

Different types of assertions are used depending upon the application.

Below is the simpler demonstration of a program that

only allows only the batch with all hot food to be dispatched,

else rejects the whole batch.

# initializing list of foods temperatures

batch = [ 40,26, 39, 30, 25, 21]

# initializing cut temperature

cut = 26

# using assert to check for temperature greater than cut

for i in batch:

assert i >= 26, "Batch is Rejected"

#print (str(i) + " is O.K" )

print (i, " is O.K" )

ex-5:

try:

x = 1

y = 0

assert y != 0, "Invalid Operation"

print(x / y)

# the errror\_message provided by the user gets printed

except AssertionError as msg:

print(msg)

**FILE HANDLING**

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

==> Variables are used to store the values temporary.

==> if we want to store the values permanently,

we go for 2 alternatives:

1. file handling

2. database

what is file ?

==> A collection of related information stored into a

one particular area on the memory.

==> That area is called a file.

==> python supports 2 types of files:

1. text files

2. binary files

1. text files:

==> These files are used to store only character data.

ex: student.txt, employee.txt,hello.txt.....etc.

2. binary files:

These files are used to store binary data.

i.e. audio,videos,images,pics,....etc.

ex: student.jpg, employee.jpeg, jntuh.png,....etc.

**file operations:**

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

==>

1. To open a file

2. To write data into a file

3. To read data from a file

4. To close the file

1. To open a file:

==> The open() built-in function is used to open a file.

==> syntax:

file object = open('file name','mode')

ex: a = open('student.txt''w')

Here,

==> file object is a file pointer, which is pointing the given file.

==> open() is a built-in function

==> 'file name' represents name of the file

==> 'mode' reprsents file opening purpose.

By default mode is reading mode.

==> The modes are:

w ---> To open a file for writing purpose.

r ---> To open a file for reading purpose.

a ---> To open a file for appending purpose.

To add new data end of the file.

w+ ---> To open a file for writing & reading purpose.

r+ ---> To open a file for appending & reading purpose.

To add new data beging of the file.

a+ ---> To open a file for appending & reading purpose.

To add new data end of the file.

x ---> To open a file for exclusive writing purpose.

2. To write data into a file:

==> python is used the following functions for writing purpose:

1. print()

2. write()

3. writelines()

1. print() :

==> this function is used to display the messages on the screen.

ex-1:

fo = open('wish.txt','w')

print('Good Morning',file=fo)

print('welcome to tronix technologies',file=fo)

print('python course',file=fo)

fo.close()

ex-2: To store student names

fo = open('names.txt','w')

print('ram',file=fo)

print('venkat',file=fo)

print('siva',file=fo)

print('laxman',file=fo)

print('sita',file=fo)

print('Data written successfully ...')

fo.close()

2. write():

==> This function is used to write the data into a file.

==> It is used for write only one line

==> syntax:

fileobject.write('Any string')

ex-1:

fo = open('names.txt','w')

fo.write('ram\n')

fo.write('venkat\n')

fo.write('siva\n')

fo.write('laxman\n')

fo.write('sita\n')

print('Data written successfully ...')

fo.close()

ex-2:

fo = open('wish.txt','w')

fo.write('Good Morning to all\n')

fo.write('tronix technologies\n')

fo.write('Near jntu signal\n')

fo.write('KPHB\n')

fo.write('HYDERABAD\n')

print('Data written successfully ...')

fo.close()

3. writelines():

==> This function is used to write the data into a file.

==> It is used for write morethan one line

==> syntax:

fileobject.writelines(' string ')

ex-1:

fo = open('names.txt','w')

fo.writelines('ram\nvenkat\nsiva\nlaxman\nsita\n')

print('Data written successfully ...')

fo.close()

write a program to store empid, ename, salary

To store only one empoyee

fo = open('emp.txt','w')

empid = int(input('Enter employee number'))

ename = input('Enter employee name')

salary = float(input('Enter employee salary'))

fo.write(str(empid))

fo.write(ename)

fo.write(str(salary))

fo.close()

write a program to store empid, ename, salary

To store n empoyees

fo = open('emp.txt','w')

n = int(input('How many empoyees :'))

for i in range(n):

empid = int(input('Enter employee number'))

ename = input('Enter employee name')

salary = float(input('Enter employee salary'))

fo.write(str(empid))

fo.write(ename)

fo.write(str(salary))

fo.write('\n')

fo.close()

write a program to store empid, ename, salary

To store n empoyees

fo = open('emp.txt','a')

n = int(input('How many empoyees :'))

for i in range(n):

empid = int(input('Enter employee number'))

ename = input('Enter employee name')

salary = float(input('Enter employee salary'))

fo.write(str(empid))

fo.write(ename)

fo.write(str(salary))

fo.write('\n')

fo.close()

**Q.**write a program to store empid, ename, salary

To store n empoyees

using while loop :

fo = open('emp.txt','w')

while True:

empid = int(input('Enter employee number'))

ename = input('Enter employee name')

salary = float(input('Enter employee salary'))

fo.write(str(empid))

fo.write(ename)

fo.write(str(salary))

fo.write('\n')

ch = input('Do you want to enter another record(y/n)?')

if ch=='n':

break

fo.close()

**Q.**write a program to store students information,

ie. htno name, and three subjects marks.

using while loop :

fo = open('student.txt','w')

while True:

htno = int(input('Enter hall ticket number'))

sname = input('Enter student name')

m1 = int(input('Enter first subject marks'))

m2 = int(input('Enter second subject marks'))

m3 = int(input('Enter third subject marks'))

fo.write(str(htno))

fo.write(sname)

fo.write(str(m1))

fo.write(str(m2))

fo.write(str(m3))

fo.write('\n')

ch = input('Do you want to enter another student record(y/n)?')

if ch=='n' or ch=='N':

break

fo.close()

write a program to store student names in names.txt file.

fo = open('names.txt','w')

fo.write('ram\n')

fo.write('venkat\n')

fo.write('siva\n')

fo.write('laxman\n')

fo.write('sita\n')

print('Data written successfully ...')

fo.close()

read the data from a file:

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

==> python supprots the following built-in functions for reading purpose:

1. read()

2. read(n)

3. readline()

4. readlines()

1. read():

This function is used to read entire data from a file.

The result is string data type.

2. read(n):

This function is used to read first 'n' characters from a file.

The result is string data type.

3. readline():

This function is used to read only one line from a file.

The result is string data type.

4. readlines():

This function is used to read entire data from a file.

The result is list data type.

1. write a program to read entire data from a file

fo = open('names.txt','r')

data = fo.read()

print(data)

print(type(data))

fo.close()

2. write a program to read first 2 characters from a file

fo = open('names.txt','r')

data = fo.read(2)

print(data)

print(type(data))

fo.close()

3. write a program to read first 3 lines from a file

fo = open('names.txt','r')

x = fo.readline()

y = fo.readline()

z = fo.readline()

print(x,end='')

print(y,end='')

print(z)

fo.close()

4. write a program to,

read entire data from a file using readlines()

fo = open('names.txt','r')

data = fo.readlines()

print(data)

print(type(data))

fo.close()

for i in data:

print(i,end='')

5. write a program to read entire data from a file

and print it upper case letters

fo = open('names.txt','r')

data = fo.read()

print(data.upper())

fo.close()

6. write a program to read entire data from a file

and print it first character is upper case letters

fo = open('names.txt','r')

data = fo.read()

print(data.title())

fo.close()

7. write a program to count No. of characters in a file

method-1: using len() built-in function

fo = open('names.txt','r')

data = fo.read()

print('No. of characters = ',len(data))

fo.close()

method-2: using for loop

fo = open('names.txt','r')

data = fo.read()

count = 0

for i in data:

count = count + 1

print('No. of characters = ',count)

fo.close()

method-3: using for loop except new line character

fo = open('names.txt','r')

data = fo.read()

count = 0

for i in data:

if i!='\n':

count = count + 1

print('No. of characters = ',count)

fo.close()

Handling csv files

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

1. CSV stands Comma Separated Values.

2. A Comma Separated Values (CSV) file is a plain text file

that contains a list of data.

3. The data separated by commas.

4. As part of our programming,

It is very common requirement to write and read data from

csv files.

5. python provides csv module to handle the csv files.

6. CSV module in Python is a built-in module that means ,

you don’t need to install it by using pip.

7. It comes with Python by default.

8. Python CSV module provides all the methods and properties

to work with CSV files.

writerow():

==> This function is used to write the data into a csv file.

==> The values are separted by commas.

==> It takes exactly only one argument.

reader():

==> The reader() function is used to return a reader object

that are capable of iterate and return

one line of given CSV file.

write a program to write data to a csv file

import csv

fp=open('message.csv','w')

w = csv.writer(fp)

w.writerow('hello')

w.writerow('good morning')

w.writerow('welcome')

w.writerow([11,22,33,44])

fp.close()

write a program to store list of employees data using csv file.

import csv

fp=open('emp.csv','w')

w = csv.writer(fp)

w.writerow([1901,'ram',56000])

w.writerow([1902,'venkat',99999])

w.writerow([1903,'siva',45000])

fp.close()

write a program to store list of employees data using csv file.

import csv

fp=open('emp.csv','w')

w = csv.writer(fp)

n=int(input('Enter Number of Employees'))

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

empno=int(input('Enter employee number'))

ename=input('Enter employee name')

sal=float(input('Enter employee salary'))

w.writerow([empno,ename,sal])

print('Total employeess data written in csv file successfully')

fp.close()

write a program to write list of student information in a file using csv file.

import csv

fp=open('student.csv','w')

w = csv.writer(fp)

n=int(input('Enter Number of students'))

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

htno=int(input('Enter student hall ticke number'))

sname=input('Enter student name')

m1=int(input('Enter first subject marks'))

m2=int(input('Enter second subject marks'))

m3=int(input('Enter third subject marks'))

w.writerow([htno,sname,m1,m2,m3])

print('Total students data written in csv file successfully')

fp.close()

write a program to read data from a csv file

import csv

fp=open('student.csv','r')

r = csv.reader(fp)

data = list(r)

print('STUDENT DATA IS :')

#print(data)

for i in data:

for j in i:

print(j,'\t',end=' ')

print()

fp.close()

**Files programs**

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

1. Write a Python program to check whether a file exists.

import os.path

open('abc.txt', 'w')

print(os.path.isfile('abcdefgh.txt'))

Note:

isfile():

This function returns eithe True/False

if given file is present in our directory, then,

it returns True.

otherwise False

2. Write a Python program to check

if a file path is a file or a directory.

import os

#path="c:/users/dell/appdata/local/programs/python/python311"

#path="names.txt"

path="names123456789.txt"

if os.path.isdir(path):

print("\nIt is a directory")

elif os.path.isfile(path):

print("\nIt is a Normal text file")

else:

print("It is a special file (socket, FIFO, device file)" )

3. Write a program to print

all three letter words in a given text file

fp = open('sample.txt','r')

x = fp.read().split()

for i in x:

if len(i)==3:

print(i)

using list comprehensions:

fp=open('sample.txt','r')

wordlist=fp.read().split()

print([word for word in wordlist if len(word)==3])

4. Write a program to print

all the words that start with 'fr' or 'tr'

fp = open('sample.txt','r')

x = fp.read().split()

for i in x:

if i[:2]=='fr' or i[:2]=='tr':

print(i)

5. Write a program Determine what percentage

of words start with a vowel.

fp=open('sample.txt','r')

wordlist=fp.read().split()

count = 0

for word in wordlist:

if word[0] in 'aeiou':

count=count+1

print(100\*count/len(wordlist))

6. Write a program to print all 7-letter words

that start with fr and end in me.

fp=open('sample.txt','r')

wordlist=fp.read().split()

for word in wordlist:

if len(word)==7 and word[:2]=='fr' and word[-2:]=='me':

print(word)

7.Write a program to print the first ten words

that start with q.

fp=open('sample.txt','r')

wordlist=fp.read().split()

i=1

for word in wordlist:

if word[0]=='t':

if i<=10:

print(word)

i=i+1

else:

break

**8.** Write a program Find the longest word that can be made using only the letters a, b, c, d, and e.

largest = 0

for word in wordlist:

for c in word:

if c not in 'abcde':

break

else:

if len(word)>largest:

largest=len(word)

largest\_word=word

print(largest\_word)

9. Python Program to Read the Contents of a File in Reverse Order

filename=input("Enter file name: ")

for line in reversed(list(open(filename))):

print(line.rstrip())

10. Python Program to Read a File and Capitalize the First Letter of Every Word in the File

fname = input("Enter file name: ")

with open(fname, 'r') as f:

for line in f:

l=line.title()

print(l)

11. wrtie a program to find the longest word in a text file

fp=open('names.txt', 'r')

words = fp.read().split()

max\_len = len(max(words, key=len))

x = [word for word in words if len(word) == max\_len]

print(x)

12. Write a program to count the No. of capital letters in a file.

With open(‘data.txt’) as fo:

count=0

text = fo.read()

for character in text:

if character.isupper():

count+=1

print(‘No. of capital letters =’,count)

13. Write a Python program to count the frequency of words in a file.

from collections import Counter

def word\_count(fname):

with open(fname) as f:

return Counter(f.read().split())

print("Number of words in the file :",word\_count("names.txt"))

14. write a program to find number of lines , words, characters in a file

method-1:

fp=open('hello.txt','r')

nl=0

nw=0

nc=0

x=fp.readlines()

for i in x:

nl=nl+1

word=i.split()

nw=nw+len(word)

for j in i:

if j!=' ' and j!='\n':

nc=nc+1

print('number of lines=',nl)

print('number of words=',nw)

print('number of characters=',nc)

method-2:

fp=open('hello.txt','r')

nl=0

nw=0

nc=0

x=fp.read()

nc=len(x)

nw=len(x.split())

nl=len(x.splitlines())

nc=nc-(nw+nl+1)

print('number of lines=',nl)

print('number of words=',nw)

print('number of characters=',nc)

15. Write a program to print palindroms in a given file:

fp=open('hello.txt','r')

l=[]

x=fp.readlines()

for i in x:

word=i.split()

for j in word:

if j==j[ : : -1]:

l.append(j)

print(l)

16. Write a program to print all palindroms with no. of times:

fp=open('hello.txt','r')

d={ }

x=fp.readlines()

for i in x:

word=i.split()

for j in word:

if j==j[::-1]:

print(d)

17. Write a Python program to get the size of a file.

import os

file\_size = os.path.getsize("abc.txt")

print("\nThe size of abc.txt is :",file\_size,"Bytes")

print()

18. Write a Python program to retrieve file properties.

import os.path

import time

print('File :', \_file\_)

print('Access time :', time.ctime(os.path.getatime(\_file\_)))

print('Modified time:', time.ctime(os.path.getmtime(\_file\_)))

print('Change time :', time.ctime(os.path.getctime(\_file\_)))

print('Size :', os.path.getsize(\_file\_))

19. #Python program to mail merger

# Names are in the file names.txt

# Body of the mail is in body.txt

# open names.txt for reading

with open("names.txt",'r',encoding = 'utf-8') as names\_file:

# open body.txt for reading

with open("body.txt",'r',encoding = 'utf-8') as body\_file:

# read entire content of the body

body = body\_file.read()

# iterate over names

for name in names\_file:

mail = "Hello "+name+body

# write the mails to individual files

with open(name.strip()+".txt",'w',encoding = 'utf-8') as mail\_file:

mail\_file.write(mail)

20. Write a program to write employee data into csv files

import csv

with open("emp.csv","w",newline='') as f:

w=csv.writer(f) # returns csv writer object

w.writerow(["ENO","ENAME","ESAL","EADDR"])

n=int(input("Enter Number of Employees:"))

for i in range(n):

eno=int(input("Enter Employee No:"))

ename=input("Enter Employee Name:")

esal=input("Enter Employee Salary:")

eaddr=input("Enter Employee Address:")

w.writerow([eno,ename,esal,eaddr])

print("Total Employees data written to csv file successfully")

21. write a program to read data from a csv files

import csv

f=open("emp.csv",'r')

r=csv.reader(f) #returns csv reader object

data=list(r)

#print(data)

for line in data:

for word in line:

print(word,"\t",end='')

print()

22. Write a program to copy the image from one file to another file

f1=open("c:/Users/Vishnu/Desktop/charith.jpg","rb")

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

bytes=f1.read()

f2.write(bytes)

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

print(f1)

print(f2)

23. Write a program to read data from numbers.txt file

# 1) Splits the text file into individual characters

# to identify the commas and parsing the individual

# tokens.

# create a list to store the inputted numbers only .

import string

numbers = list()

dataFile = open('sample.txt', 'r')

for eachLine in dataFile:

tmpStr = ''

for char in eachLine:

if char.isdigit():

tmpStr += char

elif char == ',' and tmpStr != '':

numbers.append(int(tmpStr))

tmpStr = ''

if tmpStr.isdigit():

numbers.append(int(tmpStr))

print(numbers)

dataFile.close()

24. write a program to read data from a file sample.txt,

digits stored in numbers list , alphabets stored in strings list

sample.txt is

1,

2,ram,

3,

4,sita,5,

6,7,8,laxman,10,11,12,13

solution is :

numbers = list()

strings= list()

dataFile = open('sample.txt', 'r')

for eachLine in dataFile:

substrs = eachLine.split(',',eachLine.count(','))

for strVar in substrs:

if strVar.isdigit():

numbers.append(int(strVar))

if strVar.isalpha():

strings.append(strVar)

print(numbers)

print(strings)

dataFile.close()

25. write a program to count No. of vowels in a given text file '

vowel=['a','e','i','o','u','A','E','I','O','U']

count = 0

fo = open('message.txt','r')

x = fo.read()

for i in x:

if i in vowel:

count = count + 1

fo.close()

print('No. of vowels = ',count)

26. write a prograrm to find No. of alphabets, No. of digits and symbols

na=nd=ns=0

fo = open('message.txt','r')

x = fo.read()

for i in x:

if i.isalpha():

na=na+1

elif i.isdigit():

nd=nd+1

else:

ns=ns+1

print('No. of alphabets = ',na)

print('No. of special symbols = ',ns)

print('No. of digits = ',nd)

fo.close()

**text files Extensions:**

Web standards: html, xml, css, svg, json, ...

Source code: c, cpp, h, cs, js, py, java, rb, pl, php, sh, ...

Documents: txt, tex, markdown, asciidoc, rtf, ps, ...

Configuration: ini, cfg, rc, reg, ...

Tabular data: csv, tsv, ...

binary files extensions are:

Images: jpg, png, gif, bmp, tiff, psd, ...

Videos: mp4, mkv, avi, mov, mpg, vob, ...

Audio: mp3, aac, wav, flac, ogg, mka, wma, ...

Documents: pdf, doc, xls, ppt, docx, odt, ...

Archive: zip, rar, 7z, tar, iso, ...

Database: mdb, accde, frm, sqlite, ...

Executable: exe, dll, so, class, ...

binary files

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

==> To store the binary data.

i.e. images,pic,audio,video files.....etc.

==> The binary extensions are:

Images: jpg, png, gif, bmp, tiff, psd, ...

Videos: mp4, mkv, avi, mov, mpg, vob, ...

Audio: mp3, aac, wav, flac, ogg, mka, wma, ...

Documents: pdf, doc, xls, ppt, docx, odt, ...

Archive: zip, rar, 7z, tar, iso, ...

Database: mdb, accde, frm, sqlite, ...

Executable: exe, dll, so, class, ...

==> the modes are:

wb , rb, ab, wb+,rb+,ab+

ex-1: write a program to copy the image from

one binary file into another binary file.

fo1 = open('C:/Users/DELL/Desktop/modi.webp','rb')

fo2 = open('C:/Users/DELL/Desktop/yy.webp','wb')

a = fo1.read()

fo2.write(a)

print('Image copied successfully ...')

fo1.close()

fo2.close()

ex-2: write a program to copy the image from

one binary file into another binary file.

fo1 = open('C:/Users/DELL/Desktop/ab.jpg','rb')

fo2 = open('C:/Users/DELL/Desktop/batch10.webp','wb')

a = fo1.read()

fo2.write(a)

print('Image copied successfully ...')

fo1.close()

fo2.close()

ex-3: write a program to copy the content(data)

from one file into another file.

fo1 = open('C:/Users/DELL/Desktop/demo.css','r')

fo2 = open('C:/Users/DELL/Desktop/first.css','w')

a = fo1.read()

fo2.write(a)

print('Data copied successfully ...')

fo1.close()

fo2.close()

'''

fo = open('names.txt','w')

print('ram',file=fo)

print('venkat',file=fo)

print('siva',file=fo)

print('laxman',file=fo)

print('sita',file=fo)

print('Data written successfully ...')

fo.close()

**Types of programming Langauges**

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

1. pop Langauges

2. oop Langauges

3. scripting Langauges

1. pop Langauges:

==> pop stands for procedural oriented programming.

==> A program written using Functions

ex: c,pascal,cobol,fortran....etc.

2. oop Langauges:

==> oop stands for object oriented programming.

==> A program written using classes & objects.

ex: c++,java,apex,go,move,swift,....etc.

3. scripting Langauges:

==> A program written in simple commands.

ex: javascript,vb script, ....etc.

Note:

python supports pop,oop, and scripting Features.

**object oriented programming(oops)**

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

1. classes

2. objects

3. data encapsulation

4. data abstraction

5. constructors

6. destructors

7. inheritance

8. polymorphism

**1. classes:**

==> A group of variables and methods(functions) are together

into a single unit.

==> That unit is called a class.

==> python supports 2 types of classes.

1. built-in classes

2. user-defined classes.

1. built-in classes

==> The classes which are coming from along with python software.

These are called built-in classes.

ex:

a=9

type(a)

<class 'int'>

b=1.3

type(b)

<class 'float'>

c='ram'

type(c)

<class 'str'>

d=True

type(d)

<class 'bool'>

a=[11,22,33,44]

type(a)

<class 'list'>

t=11,22,33,44

type(t)

<class 'tuple'>

s={111,22,33,44}

type(s)

<class 'set'>

x = range(1,11,1)

type(x)

<class 'range'>

y=None

type👍

<class 'NoneType'>

ex:

def display():

print('Hai i am display ....')

type(display)

<class 'function'>

2. user-defined classes:

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

==> the classes which are developed by programmer.

==> these classes are called user-defined classes.

==> syntax:

class classname:

list of variables

list of methods

ex-1: class car:

color

model

price

company name

height

width

start()

stop()

move()

park()

ex-2: class door:

color

model

price

company name

height

width

open()

close()

ex-3: class person:

color

name

surname

age

eyes

ears

noise

height

weigth

walk()

talk()

sleep()

profession()

t = 11,22,33,44,55,66 ==> t is a object of list class

b = [ ]

c = list()

**classes and objects**

what is class?

==> A group of variables and methods together into a single unit.

That unit is called a class.

==> i.e. blue print.

==> syntax:

class classname:

list of variables

list of methods

ex-1: To create empty class

class test:

pass

ex-2: To create a class only variables

class sample:

a=10

b=25

print(sample.a)

print(sample.b)

Here,

a,b are called class level variables.

How to access class members:

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

==> we can access class members by using dot operator.

==> class member means , it may be either variable/method.

===> syntax:

classname.membername

ex: test.m1()

test.m2()

test.find()

test.a

test.b

ex-3: To create a class only methods

class sample:

def m1():

print('I am method-1 in sample class')

def m2():

print('I am method-2 in sample class')

sample.m1()

sample.m2()

ex-4: To create a class with variables methods

class sample:

a=10

b=20

def m1():

print('I am method-1 in sample class')

def m2():

print('I am method-2 in sample class')

print(sample.a)

print(sample.b)

sample.m1()

sample.m2()

Here,

a,b are called class level variables

m1(),m2() are called class level methods

ex-5: write a program to create rectangle class and

find the area of a rectangle

class rectangle:

def read():

rectangle.l=int(input('Enter length value'))

rectangle.b=int(input('Enter breadth value'))

def find():

rectangle.area = rectangle.l \* rectangle.b

def displayresult():

print('Area of a rectangle = ',rectangle.area)

rectangle.read()

rectangle.find()

rectangle.displayresult()

Here,

l,b,area are called class level variables

read(),find(),displayresult() are called class level methods.

ex-6: write a program to create circle class and

find the area of a circle '''

class circle:

def read():

circle.r = int(input('Enter radius value'))

def find():

circle.area = 3.14 \* circle.r \* circle. r

def displayresult():

print('Area of a circle = ',circle.area)

circle.read()

circle.find()

circle.displayresult()

''' Here,

r,area are called class level variables

read(),find(),displayresult() are called class level methods.

what is object?

==> The physical appearance of the given class is called object.

==> we can create any No. of objects .

i.e. 'n' number of objects.

'''

class student:

def read():

student.htno=int(input('enter student hall ticket number'))

student.sname=input('enter sudent name')

student.m1=int(input('enter first subject marks'))

student.m2=int(input('enter second subject marks'))

student.m3=int(input('enter third subject marks'))

def find():

student.total=student.m1+student.m2+student.m3

student.avg=student.m1+student.m2+student.m3/3

def displayresult():

print('Total marks of the student=',student.total)

print('average marks of the student=',student.avg)

student.read()

student.find()

student.displayresult()

**what is object?**

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

==> The physical apperance of the given class is called object.

or

==> The instance of the class is called object.

==> object declaration syntax:

objectname = classname()

ex:

a=int()

b=float()

s=str()

d=bool()

c=complex()

x=list()

y=tuple()

z=set()

d=dict()

r=rectangle()

c=circle()

e=employee()

s=sample()

t=test()

p1 = person()

1. write a program to create a empty class and object

class sample:

pass

s1 = sample()

2. write a program to create a class and object .

class sample:

def m1(self):

print(type(self))

print('I am m1 ...')

s1 = sample()

s1.m1()

**Q.**write a program to create rectangle class and

find the area of a rectangle using objects

class rectangle:

def read(self):

self.l=int(input('Enter length value'))

self.b=int(input('Enter breadth value'))

def find(self):

self.area = self.l \* self.b

def displayresult(self):

print('Area of a rectangle = ',self.area)

r1 = rectangle()

r1.read()

r1.find()

r1.displayresult()

Here,

l,b,area are called object level variables.

read(),find(),display() are called object level methods

write a program to create window class and using objects

class window :

def read(self):

self.h=float(input('Enter window height'))

self.w=float(input('Enter window width'))

def display(self):

print('Window ready for delivery ...')

w1 = window()

w1.read()

w2 = window()

w2.read()

print('wait ....5 days ....')

input()

w1.display()

w2.display()

1. write a program, To create a class and object

class sample:

a=20

b=99

s = sample()

x = sample()

y = sample()

2. write a program, To create a class and object with methods

class sample:

a=20

b=99

def m1(self):

print('I am m1 ..')

def m2(self):

print('I am m2...')

s = sample()

x = sample()

y = sample()

s.m1()

s.m2()

x.m1()

x.m2()

y.m1()

y.m2()

2. write a program, To create a class and object with methods

class sample:

def \_init\_(self):

print('I am init ...')

def m1(self):

print('I am m1 ..')

def m2(self):

print('I am m2...')

s = sample()

s.m1()

s.m2()

\_init\_():

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

===> init means initialization.

===> It is a built-in function.

===> It is a special method or dunder method or

double underscore method or magic method.

==> This method is executed automatically,

at the time of object creation.

==> No need of explicit calling.

==> syntax:

def \_init\_(self):

body of the init function

ex-1:

class test:

def \_init\_(self):

print('I am init ...')

t1 = test()

ex-2:

class list:

def \_init\_(self):

self = [ ]

print(self)

a = list()

ex-2:

class dict:

def \_init\_(self):

self = { }

print(self)

d = dict()

#CLASS WITH OBJECT

#WINDOWS

class window:

def read(self):

self.l=int(input('ENTER THE LENGTH OF THE WINDOW :'))

self.b=int(input('ENTER THE BREADTH OF THE WINDOW :'))

def result(self):

print('WINDOWS ARE READY TO DELIVERY------>>>')

n=int(input('HOW MANY WINDOWS'))

print('\n')

for i in range(n):

w = window()

w.read()

print('\n')

i.result()

del statement in python:

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

==> del is a keyword.

==> This keyword is used to delete the object

from the memory.

==> syntax:

del objectname

or

del object-1,objec-2,object-3,...object-n

ex-1:

a = int()

b = float()

c = str()

d = complex()

x=list()

y=tuple()

z=dict()

s=set()

print(a)

print(b)

print(c)

print(d)

print(x)

print👍

print(z)

print(s)

del a

del b

del c

del d

del x

del y

del z

del s

print(a)

**constructors in python:**

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

==> To create a object,

it is called constructor.

==> In python, the constructor method is init() method.

==> syntax:

def init(self):

body of the contructor

==> The consturctors are 2 types:

1. default constuctor

2. parameterized constructor

1. default constuctor:

==> To create a object without parameters.

It is called default constuctor or

no arguments constructor or parameter less constructor.

ex-1:

class student:

def \_init\_(self):

print('I am default constructor ...')

s1 = student()

2. parameterized constuctor:

==> To create a object with parameters.

It is called parameterized constuctor .

ex-1:

class student:

def \_init\_(self,a,b,c,d,e):

print('I am parameterized constructor ...')

self.htno = a

self.name = b

self.m1 = c

self.m2 = d

self.m3 = e

def display(self):

print(self.htno)

print(self.name)

print(self.m1)

print(self.m2)

print(self.m3)

s1 = student(1901,'ram',77,88,99)

s1.display()

print('----------------------------------')

s2 = student(1902,'siva',88,88,88)

s2.display()

del s1,s2

demo for default constructor and destructor:

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

class student:

def \_init\_(self):

print('object created ...')

def \_del\_(self):

print('object deleted ...')

s1 = student()

del s1

class test:

count = 0

def \_init\_(self):

print('object created ...')

test.count = test.count + 1

def \_del\_(self):

print('object deleted ..')

test.count = test.count - 1

print('No. of objects = ',test.count)

t1 = test()

t2 = test()

t3 = test()

t4 = test()

t5 = test()

print('No. of objects = ',test.count)

del t3

del t4

print('No. of objects = ',test.count)

s6 = test()

s7 = test()

s8 = test()

s9 = test()

del s8

print('No. of objects = ',test.count)

'''

magic methods or double underscore methods or dunder methods:

\_\_ init \_\_ initialization

This method is exeucted at the time of object

creation.

This is executed automatically.

\_ \_ del \_ \_

This method is exeucted at the time of object

deletion.

This is executed automatically.

'''

'''

1. write a program to create empty constructor and empty destructor

class test:

def \_init\_(self):

pass

def \_del\_(self):

pass

**Q.**write a program to find the area of a circle

method-1: using parameterized constructors

class circle:

def \_init\_(self,x):

print('object created ..')

self.r = x

def find(self):

self.area = 3.14\*self.r\*\*2

def display(self):

print('Area of a circle = ',self.area)

def \_del\_(self):

print('object deleted ...')

c1 = circle(5)

c1.find()

c1.display()

del c1

method-1: using default constructors

class circle:

def \_init\_(self):

print('object created ..')

self.r = int(input('Enter radius value'))

def find(self):

self.area = 3.14\*self.r\*\*2

def display(self):

print('Area of a circle = ',self.area)

def \_del\_(self):

print('object deleted ...')

c1 = circle()

c1.find()

c1.display()

del c1

write a program to create a student class

with parameterized constructor

class student:

def \_init\_(self,hno,name,mark1,mark2,mark3):

self.htno = hno

self.sname = name

self.m1 = mark1

self.m2 = mark2

self.m3 = mark3

def find(self):

self.total = self.m1+self.m2+self.m3

self.avg = self.total/3

def display(self):

print('Total Marks = ', self.total)

print('Average marks = ',self.avg)

def \_del\_(self):

pass

s1 = student(101,'ram',77,88,99)

s2 = student(102,'venkat',88,88,88)

s3 = student(103,'balut',65,34,11)

s4 = student(104,'abc',11,22,33)

s1.find()

s2.find()

s3.find()

s4.find()

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

print('Object-1 Details ....')

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

s1.display()

print('\n\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

print('Object-2 Details ....')

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

s2.display()

print('\n\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

print('Object-3 Details ....')

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

s3.display()

print('\n\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

print('Object-4 Details ....')

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

s4.display()

'''

'''

Here,

101,'ram',77,88,99 ==> actual arguments

hno,name,mark1,mark2,mark3 ==> formal arguments

self is pointing current object.

htno,sname,m1,m2,m3 are called object level variables

ex:

class Dress:

def \_init\_(self, type, price):

self.type = type

self.price = price

def details(self):

return f"A {self.type} costs Rs.{self.price}"

d1 = Dress("shirt", 50)

print(d1.details())

d2 = Dress('Jeans',450)

print(d2.details())

constructors and destructors in python :

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

what is constructor ?

==> constructor means, To create a object of any class.

==> constructor is a special method in python.

==> The name of the constructor method is

\_init\_(self).

==> constuctor will be executed automatically at the

time of object creation.

==> The main purpose of the constructor is to

initiazlize the object level variables values.

==> per object costructor is executed only once.

==> constructor can take atleast one argument i.e. self.

==> The declaration syntax of constructor is :

def \_init\_(self):

body of the constructor

==> python supports 2 types of constructors:

1. Default constructor

2. Parameterized contructor

1. Default constructor:

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

==> It is a constructor.

==> To create a object without parameters

is called a default constructor.

ex-1:

class sample:

def \_init\_(self):

print('I am default constructor')

s1 = sample()

2. parameterized constructor:

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

==> It is a constructor.

==> To create a object with parameters

is called a parameterized constructor.

ex-1:

class sample:

def \_init\_(self,x):

print('I am parameterized constructor')

print('Hi ....',x)

s1 = sample('ram')

Write a program to find area of a rectangle,

using default constructor

class rectangle:

def \_init\_(self):

self.l = 10

self.b = 20

def find(self):

self.area = self.l\*self.b

def display(self):

print('Area of a rectangle = ',self.area)

r1 = rectangle()

r1.find()

r1.display()

Write a program to find area of a rectangle,

using parameterized constructor

class rectangle:

def \_init\_(self,x,y):

self.l = x

self.b = y

def find(self):

self.area = self.l\*self.b

def display(self):

print('Area of a rectangle = ',self.area)

r1 = rectangle(10,20)

r1.find()

r1.display()

r2 = rectangle(8,5)

r2.find()

r2.display()

Write a program to implement parameterized constructor

in a employee class

class employee:

def \_init\_(self,x,y,z):

self.empno=x

self.ename=y

self.sal=z

def details(self):

print('Employee details are :')

print('Employee Number = ',self.empno)

print('Employee Name = ', self.ename)

print('Employee salary = ',self.sal)

e1 = employee(1901,'ram',45000)

e1.details()

e2 = employee(1902,'venkat',99000)

e2.details()

e3 = employee(1903,'siva',56000)

e3.details()

Write a program to implement parameterized constructor

in a student class

class student:

def \_init\_(self,x,y,a,b,c):

self.htno=x

self.sname=y

self.m1=a

**Inheritance**

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

==> To derive the properties from old class into new class.

==> This process is called inheritance.

==> The advantage of inheritance is:

1. code reusability.

2. readability.

ex:

height

weight

age

parent --->name

surname

color

talk()

walk()

profession()

height

weight

age

child --->name

surname

color

talk()

walk()

profession()

==> The old class is called base class or super class.

==> The new class is called derived class or sub class.

==> python supports 5 types of inheritances:

1. single inheritance

2. multilevel inheritance

3. multiple inheritance

4. hierarchical inheritance

5. hybrid inheritance

1. single inheritance:

==> To derive the properties(variables & methods)

from only one base class.

It is called single inheritance.

ex: A parent

| |

| |

| |

B child

Here,

A is called base class

B is called derived class

parent class is called base class

child class is called derived class

2. multilevel inheritance:

==> To derive the properties(variables & methods)

from only one base class.

It is called single inheritance.

ex: A grad parent

| |

| |

| |

B parent

| |

| |

| |

C child

Here,

A is called base class

B is called class intermediate base class

C is called derived class

grand parent class is called base class

parent class is called intermediate base class

child class is called derived class

3. multiple inheritance:

==> The derived class is derived from multiple base classes.

It is called multiple inheritance.

i.e. morethan one base class.

ex-1:

A B C D

| | | |

| | | |

| | | |

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

|

|

|

E

ex-2:

father mother aunty uncle

| | | |

| | | |

| | | |

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

|

|

|

child

ex-3:

calc1 calc2 calc3 calc4

| | | |

| | | |

| | | |

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

|

|

|

calculator

4. hierarichal inheritance:

==> only one base and morethan one derived classes.

It is called hierarichal inheritance.

i.e.

only one base class and No. of derived classes.

ex-1:

parent

|

|

|

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

| | | |

| | | |

| | | |

child1 child2 child3 child4

Here,

parent is a base class

child1,child2,child3, child4,.... are called derived classes.

ex-2:

courses

|

|

|

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

| | | |

| | | |

| | | |

B.sc B.come B.Tech MBBS

ex-3:

B.Tech

|

|

|

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

| | | |

| | | |

| | | |

CSE IT ECE EEE

5. hybrid inheritance

==> A combination of two or more inheritances is called

hybrid inheritance.

ex:

A

|

|

|

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

| |

| |

| |

B C

| |

| |

|----------D---------------------

Inheritance declaration syntax:

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

class baseclassname:

list of variables

list of methods

class derivedclassname(baseclassname):

list of variables

list of methods

ex-1:

class A:

list of variables

list of methods

class B(A):

list of variables

list of methods

A

|

|

|

B

ex-2:

class sample:

list of variables

list of methods

class test(sample):

list of variables

list of methods

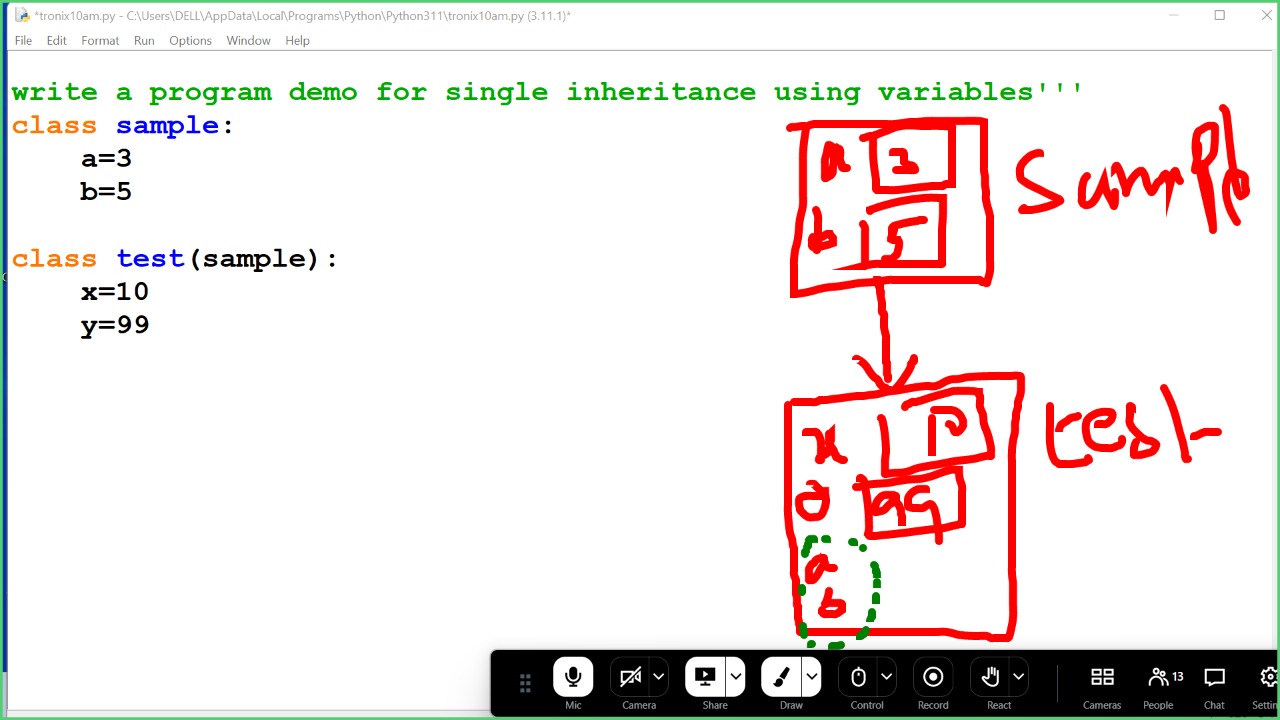
sample

|

|

|

test



Inheritance declaration syntax:

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

class baseclassname:

list of variables

list of methods

class derivedclassname(baseclassname):

list of variables

list of methods

ex-1:

class A:

list of variables

list of methods

class B(A):

list of variables

list of methods

A

|

|

|

B

ex-2:

class sample:

list of variables

list of methods

class test(sample):

list of variables

list of methods

sample

|

|

|

test

1. single inheritance:

==> To derive the properties(variables & methods)

from only one base class.

It is called single inheritance.

ex: A parent

| |

| |

| |

B child

Here,

A is called base class

B is called derived class

parent class is called base class

child class is called derived class

write a program demo for single inheritance using variables

class sample:

a=3

b=5

class test(sample):

x=10

y=99

print(test.a)

print(test.b)

print(test.x)

print(test.y)

print(test.a+test.b+test.x+test.y)

write a program demo for single inheritance using methods

class sample:

def m1():

print('I am m1 of sample class ...')

def m2():

print('I am m2 of sample class ...')

class test(sample):

def m3():

print('I am m3 of test class ...')

def m4():

print('I am m4 of test class ...')

test.m1()

test.m2()

test.m3()

test.m4()

Here,

sample is called base class or super class.

test is called derived class or sub class

write a program demo for single inheritance

class animal:

def speak(self):

print('Animal speaking ...')

class dog(animal):

def bark(self):

print('dog barking ......')

d1 = dog()

d1.speak()

d1.bark()

2. multilevel inheritance:

==> To derive the properties(variables & methods)

from another derived class.

It is called multilevel inheritance.

ex-1: A grad parent

| |

| |

| |

B parent

| |

| |

| |

C child

Here,

A is called base class

B is called class intermediate base class

C is called derived class

grand parent class is called base class

parent class is called intermediate base class

child class is called derived class

ex-2:write aprogram demo for multilevel inheritance

using the following diagram:

animal

|

|

|

dog

|

|

|

dogchild

class animal:

def speak(self):

print('Animal speaking ...')

class dog(animal):

def bark(self):

print('dog barking ......')

class dogchild(dog):

def eat(self):

print('dog child eating mutton ...')

d1 = dogchild()

d1.speak()

d1.bark()

d1.eat()

3. multiple inheritance:

==> The derived class is derived from multiple base classes.

It is called multiple inheritance.

i.e. morethan one base class.

==> syntax:

class baseclassname-1:

list of variables

list of methods

class baseclassname-2:

list of variables

list of methods

class baseclassname-3:

list of variables

list of methods

class deriveclassname(baseclassname-1,baseclaname-2,...):

list of variables

list of methods

ex-1: write a program to implement multiple inheritance.

calc1 calc2 calc3 calc4

add() | sub() | mul() | | div()

| | | |

| | | |

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

|

|

|

calculator

mod()

add(), sub(),mul(),div'''

class calc1:

def add(self,x,y):

print('Addition value = ',x+y)

class calc2:

def sub(self,x,y):

print('Minus value = ',x-y)

class calc3:

def mul(self,x,y):

print('Multipplication value = ',x\*y)

class calc4:

def div(self,x,y):

print('Division value = ',x/y)

class calculator(calc1,calc2,calc3,calc4):

def mod(self,x,y):

print('Remainder value = ',x%y)

c1 = calculator()

c1.add(11,22)

c1.sub(5,2)

c1.mul(9,6)

c1.div(9,3)

c1.mod(9,5)

in4. hierarichal inheritance:

==> only one base and morethan one derived classes.

It is called hierarichal inheritance.

i.e.

only one base class and No. of derived classes.

ex-1:

parent

|

|

|

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

| | | |

| | | |

| | | |

child1 child2 child3 child4

Here,

parent is a base class

child1,child2,child3, child4,.... are called derived classes.

ex-2: write a program demo for hierarichal inheritance

A

|

|

|

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

| | | |

| | | |

| | | |

B C D E

Here,

A is a base class

B,C,D,E are called derived classes.

class A:

def m1():

print('I am m1....')

class B(A):

def m2():

print('I am m2....')

class C(A):

def m3():

print('I am m3....')

class D(A):

def m4():

print('I am m4....')

class E(A):

def m5():

print('I am m5....')

B.m1()

B.m2()

C.m1()

C.m3()

D.m1()

D.m4()

E.m1()

E.m5()

5. hybrid inheritance:

==> A combination of two or more inheritances is called

hybrid inheritance.

or

==> A combination of single inheritance, multilevel,multiple and

hierarichal inheritances is called hybrid inheritanc.

ex-1:

A

|

m1() |

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

| |

m1() | |

B C m1()

| |

| |

|----------D---------------------

m1()

class A:

def m1():

print('I am m1 ...')

class B(A):

pass

class C(A):

pass

class D(B,C):

pass

D.m1()

ex-2: write a program demo for hierarichal inheritance

A

|

m1() |

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

| |

m1() | |

m2() B C m1()

| | m3()

| |

|----------D---------------------

m4()

m1(),m2(),m3()

class A:

def m1():

print('I am m1 ...')

class B(A):

def m2():

print('I am m2 ...')

class C(A):

def m3():

print('I am m3 ...')

class D(B,C):

def m4():

print('I am m4 ...')

D.m1()

D.m2()

D.m3()

D.m4()

class P:

a = 30

class Q(P):

b = 50

q = Q()

print(" Value of a: " ,q.a)

print("Value of b: " ,q.b)SS

Inheritance

1. write a program to demonstrate single inheritance,

class parent:

def \_init\_(self):

self.surname='nandamuri'

self.\_\_name ='ntr'

self.\_\_age=80

self.color='red'

class child(parent):

def \_init\_(self):

super().\_init\_()

self.name='balaiah'

self.age=50

def display(self):

print(self.surname)

print(self.name)

print(self.age)

print(self.color)

c1=child()

c1.display()

write a progam to demonstrate multiple inheritance.

class calc1:

def add(self,x,y):

z=x+y

print('addition =',z)

class calc2:

def sub(self,x,y):

z=x-y

print('subtraction=',z)

class calc3:

def mul(self,x,y):

z=x\*y

print('multiplication=',z)

class calc4:

def div(self,x,y):

z=x/y

print('division=',z)

class calculator(calc1,calc2,calc3,calc4):

def mod(self,x,y):

z=x%y

print('remainder=',z)

c1=calculator()

c1.add(10,20)

c1.sub(10,20)

c1.mul(10,20)

c1.div(10,20)

c1.mod(10,20)

write a program to demonstrate single inheritance

class Operations:

a = 10

b = 20

def add(self):

sum = self.a + self.b

print('Sum of a and b is: ', sum)

class MyClass(Operations):

c = 50

d = 10

def sub(self):

sub = self.c – self.d

print('Subtraction of c and d is: ', sub)

ob = MyClass()

ob.add()

ob.sub()

write a program to demonstrate single inheritance

class Employee:

x = 10

def func\_msg(self):

print('Welcome to Employee Class')

class Department(Employee):

a = 250

b = Employee.x + 22

def func\_message(self):

print('Welcome to Department Class')

def func\_changed(self):

print('old x value = ',Employee.x)

Employee.x= Employee.x + 449

print('New Value = ', Employee.x)

dept = Department()

print(dept.a)

print(dept.b)

dept.func\_msg()

dept.func\_message()

dept.func\_changed()

write a program to demonstrate multilevevl inheritance

class Addition:

a = 10

b = 20

def add(self):

sum = self.a + self.b

print('Sum of a and b is: ', sum)

class Subtraction(Addition):

def sub(self):

sub = self.b-self.a

print('Subtraction of a and b is: ', sub)

class Multiplication(Subtraction):

def mul(self):

multi = self.a \* self.b

print('Multiplication of a and b is: ', multi)

ob = Multiplication()

ob.add()

ob.sub()

ob.mul()

write a program to demonstrate multiple inheritancece.

class Addition:

a = 10

b = 20

def add(self):

sum = self. a+ self.b

print('sum of a and b is: ', sum)

class Subtraction():

c = 50

d = 10

def sub(self):

sub = self.c-self.d

print('Subtraction of c and d is: ', sub)

class Multiplication(Addition,Subtraction):

def mul(self):

multi = self.a \* self.c

print('Multiplication of a and c is: ', multi)

ob = Multiplication ()

ob.add()

ob.sub()

ob.mul()

Write a program to demonstrate hierarichal inheritance:

class person:

def getdata(self):

self.name=input('enter Name:')

self.age=int(input('enter Age:'))

self.gender=input('enter Gender:')

def display(self):

print("Name: ",self.name)

print("Age: ",self.age)

print("Gender: ",self.gender)

class student(person):

def getdata(self):

super().getdata()

self.institute=input('Name of College: ')

self.level=input('Level: ')

def display(self):

super().display()

print('Name of College: ',self.institute)

print('Level: ',self.level)

class employee(person):

def getdata(self):

super().getdata()

self.company=input('Name of Company: ')

self.sal=input('Salary: Rs. ')

def display(self):

super().display()

print('Name of Company: ',self.company)

print('Salary: Rs. ',self.sal)

s=student()

e=employee()

print('Student')

print('Enter data')

s.getdata()

print('Displaying data')

s.display();

print('Employee')

print('Enter data')

e.getdata()

print('Displaying data')

e.display()

Write a program to demonstrate hybrid inheritance :

class arithmetic:

def getdata(self):

self.num1=int(input("\nEnter the first number: "))

self.num2=int(input("\nEnter the second number: "))

class plus(arithmetic):

def add(self):

self.sum=self.num1+self.num2

class minus(arithmetic):

def sub(self):

self.diff=self.num1-self.num2

class result(plus,minus):

def display(self):

print('Sum of {} and {} ={} '.format\

(self.num1,self.num2,self.sum))

print('Difference of {} and {} ={}'.format

(self.num1,self.num2,self.diff))

r1=result()

r1.getdata()

r1.add()

r1.sub()

r1.display()

polymorphism

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

==> poly --> many

==> morphism --> forms

==> polymorphism means single name many forms

i.e single name with different types of behaviours.

son

brother

lover

husband

prasad --> student

guest

passenger

audience

customer

==> python supports 3 types of polymophisms:

1. operator based polymophism

2. function based polymophism

3. class based polymophism

1. operator based polymophism:

==> only one operator with different behaviours

It is called operator based polymophism

ex-1: using + operator

list concatenation

tuple concatenation

+ --> addition of two integers

addition of two floats

string concatenation

2+7

9

'python'+'tronix'

'pythontronix'

x=[11,22,33,44]

y=[99,88,77]

x+y

[11, 22, 33, 44, 99, 88, 77]

t=(11,22,33)

t2=(99,88,77)

t+t2

(11, 22, 33, 99, 88, 77)

ex-2: using - operator

- --> subtration of two numbers

difference between two sets.

2-7

-5

s1={11,22,33,44}

s2={33,44,55,66,77,88,99}

s1-s2

{11, 22}

ex-3: using \* operator

list repetition

tuple repetition

\* --> multiplication of two numbers

string repetition

2\*7

14

'tronix'\*10

'tronixtronixtronixtronixtronixtronixtronixtronixtronixtronix'

a=[11,22,33,44]

a\*5

[11, 22, 33, 44, 11, 22, 33, 44, 11, 22, 33, 44, 11, 22, 33, 44, 11, 22, 33, 44]

t=11,22,33

t\*4

(11, 22, 33, 11, 22, 33, 11, 22, 33, 11, 22, 33)

2. function based polymophism:

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

==> only one function with different behaviours

It is called function based polymophism

ex-1: using range() built-in function

range(1,11,1)

range(1,11)

range(6)

ex-2: using extend() built-in function

a = [11,22,33,44]

a.extend({55,66,77})

a.extend([55,66,77])

a.extend((55,66,77))

a.extend(range(1,11,1))

a.extend('tronix')

ex-3: using print() built-in function

print('good mng')

print('good mng','to all')

print(a)

print(a,b)

print(a,b,c)

print(a,b,c,d,e,f)

ex-4: using user-defined function,

to implement function based polymorphsim

def maximum(\*x):

print(max(x))

maximum(7,2)

maximum(11,22,6,8,4)

maximum(99,34,67,23)

maximum(11,22,44,66,7,8,3,5,43,3,23,233,323,33,433,34,3)

class based polymorphism:

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

==> To implement polymorphism using classes.

ex-1:

class sample:

def m1():

print('I am m1 of sample class ...')

class test(sample):

def m1():

print('I am m1 of test class ...')

test.m1()

ex-2: using \_init\_(self)

class sample:

def \_init\_(self):

print('I am init of base class ...')

class test(sample):

def \_init\_(self):

print('I am init of derived class ...')

t1 = test()

super():

==> super() function is a built-in function.

==> This function is used to exeucte super classes functions.

==> syntax:

super().functionname()

ex-3: using \_init\_(self)

class sample:

def \_init\_(self):

print('I am init of base class ...')

class test(sample):

def \_init\_(self):

super().\_init\_()

print('I am init of derived class ...')

t1 = test()

ex-4: using \_init\_(self)

class A:

def \_init\_(self):

print('I am init of class--A ')

class B(A):

def \_init\_(self):

super().\_init\_()

print('I am init of class--B ')

class C(B):

def \_init\_(self):

super().\_init\_()

print('I am init of class--C ')

c1 = C()

write a progrma demo for hierarichal inheritance using polymorphism

person

|

| \_init\_()

| display()

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

| |

| |

| |

teacher student

\_init() \_\_init\_()

display() display()

class person:

def \_init\_(self,n,p):

self.name = n

self.phno = p

def display(self):

print('Name = ',self.name)

print('Phone Number = ',self.phno)

class teacher(person):

def \_init\_(self,a,b,c):

super().\_init\_(a,b)

self.sal = c

def display(self):

super().display()

print('Salary = ',self.sal)

class student(person):

def \_init\_(self,a,b,c,x,y,z):

super().\_init\_(b,c)

self.htno=a

self.m1=x

self.m2=y

self.m3=z

def display(self):

super().display()

print('Hall ticket Number = ',self.htno)

print('First subject marks = ',self.m1)

print('Second subject marks = ',self.m2)

print('Third subject marks = ',self.m3)

s1 = student(1901,'ram',9988776655,77,88,99)

print('\t\tSTUDNET INFORMATION')

s1.display()

t1 = teacher('siva',6677889922,96000.00)

print('\t\t TEACHER INFORMATION')

t1.display()

ex: using super() function

class Animal:

def \_init\_(self):

print('Animal birth...')

class Mammal(Animal):

def \_init\_(self):

# call superclass

super().\_init\_()

print('Mammals give birth directly')

dog = Mammal()

comprehensions

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

==> simple and easy way.

1. list comprehension

2. tuple comprehension(generators)

3. set comprehension

4. dict comprehension

1. list comprehension:

==> To create a list, with simple and compact way.

It is called list comprehension.

==> syntax:

listname = [expression for loop if conditon ]

ex-1: To create a list with 1 to 10 numbers

method-1: using list() & range() built-in functions

a = list(range(1,11,1))

print(a)

method-2: using square brackets.

a = [1,2,3,4,5,6,7,8,9,10]

print(a)

method-3: using list comprehension

a = [i for i in range(1,11,1)]

print(a)

print(type(a))

ex-2: write a program to create a list even numbers between 1 to 10

method-1:

a = list(range(2,11,2))

print(a)

method-2:

a = [2,4,6,8,10]

print(a)

method-3:

a = [ ]

for i in range(1,11,1):

if i%2==0:

a.append(i)

print(a)

method-4: using comprehension '

a = [ i for i in range(1,11,1) if i%2==0 ]

print(a)

ex-3: To create a list odd numbers between 1 to 10 numbers

a = [ i for i in range(1,11,1) if i%2!=0 ]

print(a)

ex-4: To create a list square values between 1 to 10 numbers

a = [ i\*i for i in range(1,11,1)]

print(a)

ex-5: To create a list cube values between 1 to 10 numbers

a = [ i\*\*3 for i in range(1,11,1)]

print(a)

ex-6:

fruits = ['apple','mangoes','grapes','banana']

a = [i[0] for i in fruits ]

print(a)

ex-7:

fruits = ['apple','mangoes','grapes','banana']

a = [i[-1] for i in fruits ]

print(a)

2. tuple comprehension:

==> To create a tuple, with simple and compact way.

It is called tuple comprehension.

==> syntax:

tuplename = ( expression for loop if conditon )

ex-1: To create a tuple with 1 to 10 numbers

a = ( i for i in range(1,11,1) )

print(a)

print(type(a))

Note: pytyon does not support tuple comprehension.

It supports generators.

3. set comprehension:

==> To create a set, with simple and compact way.

It is called set comprehension.

==> syntax:

setname = { expression for loop if conditon }

ex-1: To create a set with 1 to 10 numbers

s = { i for i in range(1,11,1) }

print(a)

print(type(s))

ex-2: To create a set square between 1 to 10 numbers

s = { i\*i for i in range(1,11,1) }

print(s)

ex-3: To create a set,cubes between 1 to 10 numbers

s = { i\*i\*i for i in range(1,11,1) }

print(s)

ex-4: To create a set even numbers between 1 to 10

s = { i for i in range(1,11,1) if i%2==0 }

print(s)

ex-5: To create a set, negative numbers in the given list.

a = [56,-11,8,-9,-3,22,-45]

s = { i for i in a if i<0 }

print(s)

4. dict comprehension:

==> To create a dict, with simple and compact way.

It is called dictionary comprehension.

==> To create dictionary with keys & values.

==> syntax:

dictname = {expr-1:expr-2 for loop if conditon }

Here,

expr-1 represents key

expr-2 represents value.

ex-1: To create a dict with 1 to 10 numbers

d = { i:i for i in range(1,11,1) }

print(d)

print(type(d))

ex-2:

d = { i:i\*i for i in range(1,11,1) }

print(d)

print(type(d))

ex-3:

d = { i:i\*i\*i for i in range(1,11,1) }

print(d)

print(type(d))

ex-4:

fruits = ['apple','mangoes','grapes','banana']

d = {i:len(i) for i in fruits }

print(d)

ex-5:

fruits = ['apple','mangoes','grapes','banana']

d = {i:i[0] for i in fruits }

print(d)

ex-6:

#item price in dollars

old\_price = {'milk': 1.02, 'coffee': 2.5, 'bread': 2.5}

dollar\_to\_pound = 0.76

new\_price = {item : value\*dollar\_to\_pound for (item, value) in old\_price.items()}

print(new\_price)

variable length arguments:

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

==> No. of actual arguments are differ,

from one function call to another function call.

==> This process is called variable length arguments.

==> we can declare formal arguments with asterisk symbol(\*).

==> syntax:

def functionname(\*formal argument name):

body of the function

ex-1: using built-in function

print('hello') # No. of arguments are 1.

print('hello','good') # No. of arguments are 2.

print() # No. of arguments are 0.

print('hello','good','morning') # No. of arguments are 3.

ex-2: using user-defined function

def displaytuple(\*x):

print((x)

a=11

b=22

c=33

d=44

displaytuple(a) # No. of arguments are 1.

displaytuple() # No. of arguments are 0.

displaytuple(a,b) # No. of arguments are 2.

displaytuple(a,b,c,d) # No. of arguments are 4.

ex-3: using user-defined function

def maximum(\*x):

print(max(x))

maximum(6) # No. of arguments are 1.

maximum(11,122,45) # No. of arguments are 3.

maximum(77,99,23,4,7) # No. of arguments are 5.

maximum(12,56,78,334,456,22,33,54,66,8) # No. of arguments are 10 .

what is attribute?

==> It may be either variable/method within the class.

ex-1:

class sample:

a=10

def m1():

print("I am m1 ..")

print(sample.a)

sample.m1()

#sample.m2()

#print(sample.b)

Access modifiers:

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

==> python supports 3 types of access modifiers:

1. public (variables or methods)

2. protected (variables or methods)

3. private (variables or methods)

1. public:

==> By default, every attribute is public.

==> we can access from anywhere either within the class

or from outside of the class.

ex-1: To declare public variables.

class sample:

a=5

b=8

print(sample.a)

print(sample.b)

ex-2: To declare public methods:

class sample:

def m1():

print('I am m1')

def m2():

print('I am m2')

sample.m1()

sample.m2()

Here,

a,b are called public variables

m1(),m2() are called public methods

2. protected attributes:

==> The protected attributes can be declared by using

single underscore symbol( \_ ) prefixing attribute name.

==> we can access from anywhere within the class

but outside of the class only in child classes.

ex-1: To declare protected variables.

class sample:

\_a=5

\_b=8

print(sample.\_a)

print(sample.\_b)

ex-2: To declare protected methods:

class sample:

def \_m1():

print('I am m1')

def \_m2():

print('I am m2')

sample.\_m1()

sample.\_m2()

Here,

a,b are called protected variables

m1(),m2() are called protected methods

3. private attributes:

==> The private attributes can be declared by using

double underscore symbol( \_ \_ ) prefixing attribute name.

==> private attributes can be accessed only within the class.

i.e. from outside of the class we cannot access.

ex-1: To declare private variables.

class sample:

\_\_a=5

\_\_b=8

ex-2: To declare private methods:

class sample:

def \_\_m1():

print('I am m1')

def \_\_m2():

print('I am m2')

Here,

a,b are called private variables.

m1(),m2() are called private methods.

write a program to demonstrate public ,protected and private variables

class test:

a=10 #public variable

\_b=20 #protected variable

\_\_c=30 #private variable

print(test.a)

print(test.\_b)

print(test.\_\_c)

Note:

c is a private variable,

we cannot access outside of the class.

so It returns

AttributeError: type object 'test' has no attribute '\_\_c'

write a program to demonstrate public ,protected and private methods

class test:

def m1():

print('I am m1')

def \_m2():

print('I am m2')

def \_\_m3():

print('I am m3')

test.m1()

test.\_m2()

test.\_\_m3()

Here,

m3 is a private method.

It cannot access outside of class.

so It returns AttributError.

AttributeError: type object 'test' has no attribute '\_\_m3'

write a program to demonstrate public ,protected and

private variables methods

class test:

a=10

\_b=20

\_\_c=30

def m1():

print('I am m1 of public method')

def \_m2():

print('I am m2 of protected method')

def \_\_m3():

print('I am m3 of private method')

def display():

print(test.\_\_c)

test.\_\_m3()

print(test.a)

print(test.\_b)

test.m1()

test.\_m2()

test.display()

data encapsulation:

1. It is the concept of writing data(variables) and code(methods)

into a single unit.

2. you can implement of encapsulation using class.

3. It is the concept of protecting the information from

outside of access.

4. The communication between the objects must be allowed

through the functionaly only.

ex: encapsulation class

{

data:variables

code:methods

}

Rules to implement encapsulation:

1. class is public . It is visible to all the other objects

in the communication world.

2. properties are private. i.e. data must be private

3. Initialization using constructors only.

ex-1:

class test:

def init(self):

self.\_\_a=10

self.\_\_b=20

def display(self):

print(self.\_\_a)

print(self.\_\_b)

t1 = test()

t1.display()

ex-2:

class employee:

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

self.\_\_empno=x

self.\_\_ename=y

self.\_\_sal=z

def display(self):

print(self.\_\_empno)

print(self.\_\_ename)

print(self.\_\_sal)

e1=employee(101,'ram',78900.00)

e1.display()

data abstraction or data hiding in python:

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

1. Data hiding means making the data 'private', so that

it will not be accessible to the other class members.

It can be accessed only in the class where it is

declared.

2. In python, if we want to hide the variable,

then we need to write double underscore (\_\_)

before the variable name.

ex-1:

class myClass:

\_\_num = 10

def add(self, a):

sum = self.\_\_num + a

print(sum)

obj = myClass()

obj.add(20)

#print(obj.\_\_num)

Note:

The above statement gives an error because we are trying to

access private variable outside the class

ex-2:

class person:

\_\_age = 45

\_\_bankbalance = 45000000

\_\_contact = 9494494921

def display():

print(person.\_\_age)

print(person.\_\_bankbalance)

print(person.\_\_contact)

person.display()

Datatypes in python

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

int

float

str

complex

bool

list

tuple

set

dict

frozenset

bytes

bytearray

None

range

ex:

int --> only digits

float --> digits with decimal point

str --> alphabets, numbers,special symbols

complex --> a+bj format

bool --> Either True/False

list --> Group of values ---> [ ]

tuple --> Group of values ---> ( )

set --> Group of values ---> { }

dict --> Group of key-value pairs ---> { }

frozenset

bytes

bytearray

None

range

what is frozenset?

==> frozenset is similar to set.

except, it is Immutable.

==> insertions and deletions are not allowed.

ex-1:

s = {11,22,33,44,55}

print(s)

print(type(s))

fs = frozenset(s)

print(fs)

print(type(fs))

ex-2:

s = {11,22,33,44,55}

fs = frozenset(s)

print(fs)

fs.add(99)

print(fs)

ex-3: To create a empty frozenset

x = frozenset()

print(x)

print(type(x))

ex-4: To create a frozenset using range() '

x = frozenset(range(1,11,1))

print(x)

print(type(x))

ex-5: To create a frozenset

x = frozenset([11,22,33,44,55,66])

print(x)

print(type(x))

Iterators in python

===================

==> Iterator is an object that can be iterated upon,

on the top of the iterator object, we

can apply the iterations.

==> we can create iterator object of any class(list,tuple,set,dict),

that class should contain two methods:

\_iter\_()

\_next\_()

==> We can get the iterator object of any class object

by calling iter() function.

==> If we want to get one by one elements

from iterator object by calling next() function.

==> If no more elements in our iterator object,

still we can call next() function

to raise StopIteration exception.

==>

ex-1: To create a iterator object

x=[11,22,33,44]

i = iter(x)

print(i)

print(type(i))

ex-2: To print list elements using iterator object.

x=[11,22,33,44]

print(type(x))

i=iter(x)

print(type(i))

print(next(i))

print(next(i))

print(next(i))

print(next(i))

print(next(i))

print(next(i))

Note:

iter()---> This function is used to create a iterator object.

next()---> This function is used to return next value.

ex-3: To create a class and use iter() ,next() within the class

class sample:

def \_init\_(self):

print('Constructor is executed')

print('class Object created')

def \_iter\_(self):

print('Iterator object is created')

return self

def \_next\_(self):

print('Next function is executed')

s1=sample()

i=iter(s1)

next(i)

next(i)

next(i)

next(i)

ex-4: write a program to print the power of 'n' values.

class powertwo:

def \_init\_(self,m):

self.max=m

def \_iter\_(self):

self.n=0

return self

def \_next\_(self):

if self.n < self.max:

result=self.n \*\* 2

self.n=self.n+1

return result

else:

raise StopIteration

p1=powertwo(10)

i=iter(p1)

print(next(i))

print(next(i))

print(next(i))

print(next(i))

print(next(i))

print(next(i))

print(next(i))

print(next(i))

print(next(i))

print(next(i))

print(next(i))

Generators in python

====================

==> Generator is a function, which contains one or more

yield statements.

==> By Yield keyword is also like as return statement

in a normal function.

normal function generator function

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

def f1(): def f1():

---- ----

---- ----

---- ----

return yield

f1() f1()

==> whenever return statement is executed, the control

transfer to end of the function.

==> whenever yield statement is excuted, the control transfer

to main program.

==> The generator function return the generator object.

==> whenever you are calling a next function,

the generator function must be executed upto yield keyword.

==> if no more yield statements in our generator function,

still you call generator function, to raise

StopIteration exception.

ex-1:

def f1():

print('Hai')

print('Good morning')

yield

print('welcome to kosmik technologies')

print('python class....')

yield

print('welcome to hyderbad')

print('welcome to India')

yield

g=f1()

print(type(g))

next(g)

next(g)

next(g)

ex-2:

method-1 : without generator usig def keyword

def display():

print('You are entered into display function')

return

print('I am display function...')

return

print('You are exit from display function')

return

display()

display()

display()

method-2: using yield keyword with generator function

def display():

print('You are entered into display function')

yield

print('I am display function...')

yield

print('You are exit from display function')

yield

a = display()

print(type(a))

print(a)

next(a)

next(a)

next(a)

next(a)

ex-3: to print the values...

def f1():

n=1

print('Hai')

yield n

n=n+1

print('Hello.....')

yield n

n=n+1

print('good Morning')

yield n

g=f1()

print(type(g))

a=next(g)

print(a)

b=next(g)

print(b)

c=next(c)

print(c)

or

def f1():

n=1

print('Hai')

yield n

n=n+1

print('Hello.....')

yield n

n=n+1

print('good Morning')

yield n

g=f1()

print(type(g))

print(next(g))

print(next(g))

print(next(g))

ex-4: demo for yield keyword

def f1():

yield 10

yield 20

yield 30

g=f1()

print(next(g))

print(next(g))

print(next(g))

ex-5: To print 5 to 1 numbers

method-1: using for loop

for i in range(5,0,-1):

print(i)

method-2: using while loop

i=5

while i>0:

print(i)

i=i-1

method-3: using ordinary functions

def countdown(n):

print('start countdown')

while n>0:

print👎

n=n-1

countdown(5)

method-4: using generator functions

def countdown(n):

print('start countdown')

while n>0:

yield n

n=n-1

values = countdown(5)

print(type(values))

for i in values:

print(i)