**Dt- 16-02-23**

# Features of python

### -simple

python is simple programming language

when we read python program, we feel like reading English sentence

hence understanding & developing programs become easy

### -easy to learn

the structure of python program is very simple

python uses very few keywords

hence learning & developing programs in python become easy

### -open source

python is freely available from www.python.org can be downloaded & install

there is no need of paying anything

### -high level language

programming languages are of two types: low level & high level

a low level language uses machine code instruction to develop a program

python us high level programming language as it uses english word to develop a program

### -dynamically typed

in python we not need to declare

and assignment statement binds a name to object an object can be of any type

### -platform indipendent

after the compilation of program it generates a byte code

the byte code is fixed set of instruction that runs on all operating system

### -portable

as we know python code are platform indipendent, its code will give same result on the any computer in the world

Execution of python program

1. suppose that we have a python program named first.py
2. here 'first' is the program name and .py is extension
3. after writing the code the next step is to compile the code
4. the compiler will convert the python program into the byte code
5. byte code represents a fix set of instruction that represents all operations, which run on any operating system and hardware
6. it means bytes instructions are system indipendent
7. the computer cannot execute the byte code so it is necessary to convert byte code to machine code
8. PVM(Python Virtual Machine) interpretes the byte code & convert it to the machine code

-normally after compileing the code .pyc file generate that is converted in the machine code using PVM

-but we can not see the .pyc file becase it is done internally

-if you want to create we must use following command

**python -m py\_compile file.py**

-m option in the above command represent the module of py\_compile

-this will generate .pyc file for us

-cpython name in the .pyc file indicats that we are using python compiler that was created using c lang

-the compiler created by c language is consigered as standard compiler

-so you can run the .pyc file using the name with extension .pyc

viewing the byte code

**python -m dis file.py**

1)cpython- compiler which is developed using c-lang(standard)

2)pypy- compiler which is developed using python

python uses compiler(1) + interpreter(2)

**Dt - 17-2-2023**

**PVM (Python Virtual Machine)**

-since computer understand only machine code it is ovious that we should convert any program

into machine code before it is submitted to the computer for execution

A compiler normally converts the program source code into machine code

python compiler does the same task but in a slightly different manner

it convert the program source code into byte code(in the .pyc format)

the role of pvm is to convert the byte code instructin into machine code

so that the computer can execute those machine code instruction in final output

a PVM is eqipped with interpreter which converts the byte code into machine code

and sents it to the computer processer for the execution

since an interpreter plays import role often a PVM is also know as interpreter

**in python prompt**

*help()*

var1, var2=10, 'abc'

print(var1,var2)

print("{1}Hello {0}".format{var1,var2})

print("{1}Hello %10s{0}".format(10,20))

**Dt -20 -2-2023**

# Memory Management

-in the programming languages like c, c++ programmers needs to allocate & deallocate memory during runtime

-in python allocation & deallocation are done automatically

-python's PVM will take care of such issues

-a python considers string, list, function and even modules as an object

-the memory manager inside the PVM allocates memory required for object created in a python program

-all the objects are stored in the seperate memory called a heap

-heap is memory which is allocated during runtime

-the size of heap memory depends on the ram which can increase or descrease its size depending upon the size of the program

-the OS will help to allocate the memory in the ram

-a raw memory allocator look after whether memory is available for storing objects

-on the top of raw memory allocator there are several object-specific allocators which operates on the same heap

-this object-specific allocator do the task of memory management depending upon the type of objects

# **Comparison between Java VS Python**

**java**

-java is oop languages, functional programming features are introduced in to java 8.0

-java code requires more number of lines

-it is composary to declare the datatypes of variables, array etc in java

-java has do..while, while, for and for...each loops

-java has switch statement

-memory allocation is done by automatically by JVM

-java supports single and multi dimensional arrays

-the array index should be positive integer

-a semi-color is used to terminate the statement and comma is used to seperate the expressions

**python**

-python has the functional programming with object oriented features

-python programs are concise & compact

-type declaration is not required in python

-python has while & for loops

-python does not have switch statement

-memory allocation is done by automatically by PVM

-python supports only single dimensional arrays, to work with multi-dimensional array we should use third party application like 'numpy'

-the array index can be positive or negative integer number. negative index represent location from the end of the array

-new line indicates end of the statements and semi-colon is used as expressions seperator

**Dt - 21-2-2023**

# **Comments in python**

**there are two types of comments in python**

1. single line comment

2. multiline comment

### **single-line comment**

single line comments use # at the starting of line

the entire line after the hash symbol will be considered as a comment

**ex:**

the below line will print atmiya university

print('atmiya university')

print('rajkot') #rajkot will be print

comments are not-executable statements

it means nither the python compiler nor the PVM will execute them

### **multi-line comment**

when it is required to mark more then one line as a comment,

writing hash sign in the beginneing of the every line is tedious

instead of writing hash at a starting of every line we can use (""") or (''')

before & after the lines to be commented

# **Docstrings**

in fact python supports only single line comment

multi-line comments are not available in python

the tripple double quotes and tribble single quotes are actually not multi-line line commenst but they are regular string with the exception that they can span multiple lines

in other words memory will be allocated to this strings internally

if this strings are not assigned to any variables, then they are removed from the memory by the garbage collector and hence this can be used as comments so """ or ''' is not recommended to be used since they internally occupy memory and would wast the time of the interpreter since the interpreter has to check them

**//comments are not ignored in docstring**

if we write a string inside the """ or ''' as first statement in a module, function, class or a method then this strings are called documentation string or docstrings

this docstrings are useful to create and API documentation file from a python program

an API documentation file is text file or html file that contains description of all the features of software, language or a product

this file describe all the classes , modules functions etc which are written in the software

so we can understand the api documentation file is like a help file for the end user

lets take an example how to tak an API

#use of function

def add(x,y){

"""

this function takes 2 numbers and add them

the result of addition will be displayed

"""

print("the sum of two numbers are"(x+y))

}

#calling the function

add(3,4)

we assume that the name of this file is

inorder to create an api file we need to create API files like

python -m pydoc -w docstring.py

**Dt - 22-2-2023**

# **Garbage Collection in python**

garbage collection is a module in python that is useful to delete objects from memory which are not used in the program

the module of garbage collection in python is known as "GC"

garbage collector maintain the count of each object regarding number of times it is refered(used)

when an object is used three times then its reference count would be three

count greater then zero means an object is being used or refered

garbage collector runs automatically

but in the case when more number of objects is created and if the system runs out of memory when automatic garbage collector will not run

in this case manually garbage collection is to be used

**manually garbage collection is of two types**

1. time based manual garbage collection

2. event based manual garbage collection

time based manual garbage collection: if the garbage collector is called in sertain interval of time, (recuring) it is called time based manual garbage collection

event based manual garbage collection: the garbage collector is called on the bases of event when user disconnects from an application it is called event based garbage collection

if garbage collector called frequiently, it will slow down the program execution

# **How python SEES variable**

in programming languages link c & java the concept of variable is connected to the memory location

this kind of programming languages uses variable as a storage box

suppose that we write a=7 then 'a' is allocated some memory and 7 is stored into it

now suppose we change the value of 'a', a=34 then it may be visualized as below

when we assign one variable to another variable then it may be visualized as (b=a)

but on python variable is seen as a tag(name) that is tied to some value

for example a=7, in python it would be link "a->7"

now suppose we change the value of 'a', a=34 then the tag is simply changed to the new value

assigning one variable to another (b=a) makes a new bound to the same value

python has "tags" to represent the values

python is using memory efficiently

**Dt -23 -2-2023**

# **Date Type in python**

an integer data type represents an integer number.

the number must be without decimal or fraction values

values can be either positive or negative

### **integer datatype**

it can store very large integer value easily

### **float datatype**

a float datatype represents a floating point number

the number must be decimal or fractional value

it also accepts number in scientific notation where we use 'e' or 'E' to represent the power of 10

3.4\*(10)3 is written as 3.4e3

suppose we write like a=3.4e3 then 3.4\*(10)3 is stored in to variable a

### **Complex datatype**

real number + imegenary number

a complex number is number that is written in the form of a+bj

here 'a' represents the real part or number and 'b' may represent imegenary part of the number

the part 'a' of 'b' may contain integer of float value

a=2.5+2.5j

b=3.6+-9.6j

c=a+b

print(c)

representing binary, octal and hexadecimal numbers

a binary numbers are written by prefixing 0b(zero and b or B) before the value

0b1100110001

octal numbers are written by prifixing by

0O(zero and o or 0)

0o214

and hexa decimal number are written prefixing by

0x(zero and x or x)

oxd214

no1=0b01011110110

no2=0o546

no3=0xd65dfdf

i=int(no)

print('the decimal of \_\_\_\_\_ is', i )

#the other way to do same thing as above

no1=1011000110

no2=214

no3=214

i=int(no1,2)

print('the decimal of 101110011' is ', i)

i=int(no1,8)

i=int(no1,16)

#we can convert a number to binar using bin(), to octal using oct() and to hexadecimal

no=7

binary=bin(no)

print(binary)

bin()

oct()

hex()

## **Converting the datatype explicitely**

python assumes the datatype for the variable depending on the type of data

if somebody wants to convert the datatype by his own, it is called type convertion

boolean data type

no=7<34

print(no)

print(True + True)

print(True + False)

print(False + True)

print(False + False)

print(True - True)

print(True - False)

print(False - True)

print(False - False)

**Dt -24-2-2023**

## 1. sequences

sequence represent group of elements or item

a group of integer number will form a sequence

multi line comment is not actual comment it is juggad

'''multi line comment is treated as string and it consume memory '''

a string is represented by a group of character which are enclose in single quote or double quote

a="atmiya"

b='university'

string can also be written inside tripple double quote or tripple single quote to span group of lines including spaces

sqt='''string can also be written inside tripple double quote or tripple single quote to span group of lines including spaces'''

sst="""string can also be written inside tripple double quote or tripple single quote to span group of lines including spaces"""

print(a[0])

print(a[11:22])

print(a[-1])

print(a[-12:-6])

print(a\*2)

to retrive peace of string from the whole string

square breakets are used

the count of string begin with zero

sqt='''string can also be written inside tripple double quote or tripple single quote to span group of lines including spaces'''

sst="""string can also be written inside tripple double quote or tripple single quote to span group of lines including spaces"""

## 2. bytes datatype

**0-255 positive numbers**

a=[2,7,11,45,48] this is list of bytes number

b=bytes(a) convert thel ist into bytes array

the bytes datatype represents a group of byte numbers just line an array

a byte number can be any positive numbers between 0-255(both included)

we can not modify any element from bytes

'bytes object doesnot support item assignment'

b[a]=8 #this is not allowed in byte

**bytearray datatype**

bytearray()

bytearray datatype is similar to bytes datatype, with one difference

the difference is that we can modify the elements of byte array

b[a]=8 #this is allowed in bytearray

**byte vs bytearray**

## 3. list datatype

list in python is similar to arrays in c or java

a list represent a group of elements

the difference between list and array is that a list can store different types of elements but array can store only one type of elements

the size of array is fixed but list can grow dynamically

elements of list are written between [] breakets

elements of list can be modified

## 4. tuple datatype

a touple contains a group of elements which can be of different types

list and tuple are same with two differenc the first differenc is that elements of tuple are written between parenthes

and the second differenc is elements of tuple can not be modified

#range datatype

a=range(8)

for i in a:print(i)

a=range(11, 44, 2) #it will generate range starting from 11 with 1 jump everytime till 43

the range datatype represent a sequence of number the numbers in the range are not modifiable

generally range is used for repeating a for loop

for a specific number of times

a=range(11,44,2)#

for i in a:print(i)

**Dt -25-2-2023**

## 4. sets

set is case sensitive

'A' and 'a' are totaly different

list to set

tuple to set

a set is an unordered collection of elements in which order of elements is not maintain

it means the order in which the output is generated may not be same as the input

a set does not accept the duplicate elements

we can add or remove elements to/from

we cannot retrive any element from the set as it does not generate set in the same order

set datatype

elements in the set must be enterd, between curly breackets

#creating the set

a={12,3,6,3,63,-25}

print(a)

while we observe the output generaged we can see that the sequence in which elements were entered in the set are changed when we print the output

the other significant things to observe is that we have entered 34 two times but when we printed the set 34 appeared only for 1 time

we can use set functin to create a set

#creating the set usint set()

a=set("Atmiya")

print(a)

a=set("AAtmiya")

print(a)

we can convert list to set as below

#list can be converted to set using set()

a=[,2,2,41,-1,2] #list

print(a)

a=set(a) #set

print(a)

we can add elements to the existing set using update function

#adding elements to the set

a.update([10,20])

a.update(10,20)

print(a)

we can remove elements from the set using remove function

#removing the element from the set

a.remove(10)

print(a)

## Frozenset datatype

## set vs frozenset

the frozenset set datatype is same as the set datatype with one difference that data cannot be modified

we can create the frozenset by passing a set to the frozenset() function

#conterting the set to the frozenset

a={10,20,3,-8,96,78}

fs=frozenset(a)

print(fs)

#mapping types(dictionary)

a map represents group of elements in the form of key and value pairs

while retriving the value, the value retrived would be the one which is associated with the key

the key and its associated value should be seperated by a colon and every pair must be seperated by a comma

all the elements must be enclosed inside curly breackets

a={'name':'jenish', 'surname': 'ginoya'}

b={1:'abc', 2:'pqr', 3:'jkmn', }

#retriving the value using [key] as below

print(a['name'])

#retriving all the keys

print(a.keys)

#retriving all the values

print(a.values)

#updating the value of key

a['name']='jk'

#deleting the element

del is keyword

del a['name']

#we can create empty dictionary

a={}

#we can create empty set

a=set()

#adding keys & values to the empty dictionary

a[1]='abc'

a[4]=15

a['ab']='jd'

st={52,5,6,3,6,9,6-1,4, -89}

#print(st)

a=set("Atmiya")

print(a)

**Dt -27-2-2023**

# Literals in python

a constant value stored in the variable in a program is known as literals

a=25

here 'a' is the variable in which constant 25 is stored, 25 is called literal

as 25 is an integer value it is called integer literal

**types of literals found in python are as below**

- numeric literals

- boolean literals

- string literals

### **1. numeric literals**

available numeric literals in python are as below

ex literal name

1. 3.22, -3.22, 1.24e7 - float literals

2. 0X25a -hexa decimal literals

3. 0o1203 -octal literal

4. 0b01110111 -binary literal

5. 16+7j -complex liternal

6. 340, -34 -integer literal

### **2. boolean literal**

boolean literals are in the form of "True" or "False" values stored in boolean type variable

### **3. string literal**

a group of characters is called a string literals

this string literals are enclosed in single quotes or double quotes or sometimes triple quotes as required

in python there is no difference between single quoted strings or double quoted string

a='Atmiya'

a="Atmiya"

a='''atmiya

uni-rajkot'''

a="""atmiya

uni-rajkot"""

# **Escape characters**

a=''' atmiya university \

"yogidham" \nkalavad road-rajkot '''

a=''' atmiya university \\ "yogidham" \nkalavad road-rajkot '''

we can use escape characters as required in a string literal

escape character meaning

1. \ -new line continuation

2. \n -new line

3. \\ -display single backslash

4. \' -display single quote

5. \" -display double quote

6. \b -display backspace

7. \t -display horizontal tab

8. \r -enter

9. \v -vertical tab

## **determining the datatype of a variable**

we can use the type() function to know the datatype of the variable of object

intg=10

print(type(intg))

since we are storing since 10 into a is assume by the python interpreter as 'int'

it means that variable a is an object of the class int

infect every datatype function, method, class, module, list, sets, are all objects in python

flt=21.3

print(type(flt))

strg="atmiya"

print(type(strg))

lt=[44,-96,633.3]

print(type(lt))

tpl=(44,-96,633.3)

print(type(tpl))

st={44,-96,633.3}

print(type(st))

#characters

python does not contain 'char' datatype to represent individual character it has string(str) datatype which represent strings

a='A'

print(type(a))

we can access the individual characters of a string using index or position number

a="Atmiya"

print(a[0])

**Dt -28-2-2023**

# **constant in python**

CAPITAL

a constant is similar to a variable but its value cannot be modified during program execution

value of normal variable can be change when required, which is not possible for a constant

once define a constant value is not allow to change

ex

pi=22/7 or 3.14

"pi" in mathematics is 22/7 which enable changes

python doesnot allow to creating constant value

a programmer can indicate a variable as constant by writing its name in capital letters

MAX\_VALUE=50 but its value can be chaged

#identifere

-an identifere is name that is given to a variable, function or a class & object

-identifere can include letters, numbers & underscore

-identifere should not start with numberc values

-use of other special character is not allow. a python is case sensitive language

a=90

A=80

this both variable are totally different

Write down from the classroom image

**#Reserved Words**

**#Naming convention**

**Dt -01-03-2023**

## pip install playsound

pip - python install package

**Dt 04--03-2023**

# Operator in python

an operator is symbol that performs an operation an operator acts on same variables which are known as operator we cal classify operator depending on the nature of the variables

### **-arithmetic operators(7)**

this type of operator are used to performs simple arithmetic operations

+ - addition operator. adds two values

- - subtraction operator. subtracts one value from another

\* - multiplication operator. multiplies two values

\*\* - exponentiation operator. calculates exponentiation power value

/ - division(float) operator. divides left operand by the right operand

// - division(int) - integer division. performs division and gives only integer quotient

% -modulus

### **-assignment operators()**

generally this type of operator are used to store the right side value to the left side variable

myvar=12

we can also use this operators to perform some simple arithmetic operations and store it to the variable

it is possible to assign the same value to two variables

a=b=45

a,b=45

it is also possible to assign different value to two variables

a,b=10,20

shorthand operator

= a=b+c

a=a+b or a+=b

a=a-b or a-=b

a=a\*b or a\*=b

a=a\*\*b or a\*\*=b

a=a/b or a/=b

a=a//b or a//=b

a=a%b or a%=b

### **-unary minus operators**

the unary minus operator is denoted by '-'(minus), when the operator is used to before the variable

by doing so the value of the variable is negated. it means if the value of the variable is positive then it will conterted to negative and if its value is negative then it will converted to positive

#negation

a=10

print(-a)

a=-50

print(-a)

### **-relational operators**

relational operator used to compare two values or variables, whether values in both variables are same or which one is bigger or which one is lesser.

this operators give result into true or false

a=10

b=15

a>b

a<b

a>=b

a<=b

a==b

a!=b

**Dt -06-03-2023**

### **-Logical operator**

logical operator are useful to construct compound conditions

a compound conditions is a combination of more than one simple condition

each of the simple condition eveluated to or False and then the decision is

taken to know whether the total condition is true or False

in case of logical operators False indicates 0 and True indicates any other number

### **-boolean logical operator**

the result provided by boolean operators will either True or False

**Dt -07-03-2023**

### **-Membership operators**

-in

-not in

flowers=["mogra", "roase", "sunflower", "marigold"]

the membership operators are useful to test for membership in a sequence such as in a sequence string list tuptle or dictionary.

for example if an element is found in the sequence or not can be asserted using this operators.

there are to membership operators:

-in

-not in

#### **The "in" operator**

this operator returns True if an element is found in the specified sequence.

if the element is not found in the sequence, then it is returns False.

#### **The "not in" operator**

this operator returns true if an element is not found in the specified sequence

if the element is found in the sequence, then it returns False

### **-identity operators**

this operator compare the memory location of two objects.

the memory location of the objects can be seen using the id() function

this function returns an integer number, called identity number that internally represents the memory location of the objects

Note: this do in IDLE

identity number= memory address

there are two identity operators

-is

-is not

#### **the 'is' operator**

the 'is' operator is used to compare whether two objects are located at the same location in the memory or not

it will internally compare the identity number of the objects

if the identity number of the objects are same it will returns True otherwise it will retur False

#### **the 'is not' operator**

if the identity number of the objects are not same it will return True otherwise it will retur False

**Dt -10-03-2023**

# input and output statements

# the data given to this computer is called input. the result return by the computer are called output. so we can say that a computer take input and process that input and produce the output

#output statements

# the output in python is displayed using the print function. the print function can be used in different formats

#the print() statements

# when the print function is called, simply it will throught coursour to the next line it means a blank line will be displayed

# the print("string") statement

# the string represent a group of characters when a string is passed to the print function, the string is displayed as it is

# EX:

print('atmiya')

# NOTE: note that single quotes or double quotes have the same mining

# we can use escape sequences characters inside the print function

print("this is a \n new line")

print("the tab is \t is used here ")

# the plus operator is used with number will add both numbers but when used with strings it will works as concenation operator. it will jon string

print("the city is " + "Rajkot ")

#the print(variables list) statements

# we can also display the values of the variables using the print function

a,b=7,16

print(7,16)

# we can seperate the string using 'sep' attribute

print(7,16, sep=",")

print(7,16, sep="-")

# each print function throught the function to the next line after displaying output

print('atmiya')

print('university')

print('rajkot')

# we can use "end" attribute as below

print('atmiya', end="\t")

print('university')

print('rajkot')

#the print(object) statement

a=[7, 'atmiya', 'rajkot'];

print(a)

#the most common use of the print function is to use strings along with variables inside the

print function

#the print("string", variables list) statement

a=7

print(a, "the value is printed here")

print("user have entered:", a," as an input")

#the print(formatted string) statement

# we can fromat the output using the print function. for that purpose, the special operator '%' is used to joint string with a variable. we can use %i, or %d to represent decimal integer number we can use %f to represent float value we can use %s to represent string

a=8

print("value is %i"%a)

a,b=10,16

print("a is %i b is %i "%(a,b))

a,b=10,16

print("a is %i b is %i "%(b,a))

university="atmiya"

print('hi %s'%university)

print('hi %20s'%university)

print('hi (%-20s)'%university)

print('the first character is %c, and the second character is %c'%(university[0], university[1]))

num=1254.3363222

print('the num is %f'%num)

print('the num is %6.2f'%num)

#total 6, in them 2 decimal

#inside the formatted string

no1, no2, no3=1,2,3

print("the first number is {1}, the second number is {0}, the third number is{2}".formate(no2, no3, no1))

**Dt -13-03-2023**

# **Command Line Arguments**

we can write a program which can accept the values while we are running our program in the command prompt

command line arguments are the arguments which are passed to the program from the outside the program artuments must be entered by keeping white space between them

the arguments passed at the command line are by default stored in the form of list

with the name 'argv' which is the part of 'sys' module

import sys

print(sys.argv)

a=sys.argv

print(a)

print(('the file name is :',a[0]))

print(type(a))

print(id(a))

write a program to accept the two value from the user at the command line and add them

a=int(sys.argv[0])

a=int(sys.argv[1])

b=int(sys.argv[2])

sum=a+b

print('the sum of a & b is: ', sum)

print("Address of sys var: ",id(sys.argv))

print("Type of sys var: ",type(sys.argv))

command line arguments with 'argparse'

the 'argparse' module in python is useful to develop user friendly programs using command line arguments

the 'argparse' module generates help and usage messages when user gives the program invalud arguments

it may also display an error messages to the users

to work with 'argparse', we must import it as 'import argparse'

if 'argparse' is not installed then install it using command 'pip install argparse' in the command prompt

to work with argparse module it is necessary to import it as

'import argparse'

the we should create an object of 'ArgumentParser' class object with description of the program

import argparse

parser=argparse.ArgumentParser(description='enter any integer value, program will give you the square of it.')

the next step is adding arguments to the parer using add\_argument() method.

a program may have one or more arguments that are to be inputed by the user at the time of running the program

parser.add\_argument("no", type=int, help='please insert one integer number')

in the above statement "no" represents the variable where arguments is stored

type=int means user can enter only integer value

help represents help message displayed to the user

the parser will parse(go through) the arguments provided by the user

this arguments are received by pars\_args() method as

args=parser.parse\_args()

write a program to find the square of a given number using argparse in command line

import argparse

parser=argparse.ArgumentParser(description='enter any integer value, program will give you the square of it.')

parser.add\_argument("no", type=int, help='please insert one integer number')#here no is variable

args=parser.parse\_args()

ans=args.no\*\*2

print('the square of the inserted number is : ',ans)

parser.add\_argument("no1", type=int, help='please insert one integer number')

parser.add\_argument("no2", type=int, help='please insert one integer number')

**Dt -16-03-2023**

program to add two number using command line arguments and argparse

import argparse

parse=argparse.ArgumentParser(description="enter two integers numbers, program will give you same of it")

parse.add\_argument('n1', type=int, help="pleac insert ")

program to assept multiple values from the user

parse=argparse.ArgumentParser(description="enter values as required : ")

parse.add\_argument('val', nargs='\*')

args=parse.parse\_args()

for a in args.val:

print(a)

nargs='+' -> here minimum 1 value required

nargs='\*' -> here 0 value is possible

nargs='1'

'nargs' keyword

**Dt -20-03-2023**

# unit-2

## Looping and ControlSstructure, Arrays, Strings

# if statement

the if statement is used to execute one or more statements depending upon the wheter condition is true or not

syntax:

if condition:

statements

first the condition is tested. if the condition is true then the statements given after colon are executed

we cṇan write one or more statements after colon. if the condition is false then the statements given after colon are executed mensioned after colon are not executed

if a==1:

print ('the value of a is: 'a)

a=int(input('enter the value of a: '))

if a>7:

print('the entered value is greater then 7')

print('thank you')

a=int(input('enter the value of a: '))

if a>0:

print('the value you entered is positive')

a='atmiya'

if a=='atmiya':

print('the value of a is Atmiya')

a=int(input('enter the value of a: '))

b=int(input('enter the value of b: '))

if a>b:

print('the value of a is greater than b')

a=int(input('enter the value of a: '))

b=int(input('enter the value of b: '))

if a==1:

print('the value of a is greater than b')

if b==2:

print('the value of b is 2')

print('end of program')

quantity=int(input('enter the quantity: '))

price=5

if price\*quantity<10000:

print('the bill amount is less than Rs. 10000, we cannot dispatch your order')

print('pleace increase your quantity')

### if...else statement

the if else statement executes a group of statements when a condition is true otherwise it will execute another group of statements

syntax:

if condition:

statement 1

else:

statement 2

a=int(input('enter the value of a: '))

if a%2==0:

print('the enter number is an even number')

else:

print('the enter number is an odd number')

a=int(input('enter the value of a: '))

if a==1:

print('the value of a is 1')

else:

print('the value of a is other than 1')

a=input('enter the value of a: ')

if a=='atmiya':

print('user has entered atmiya as input')

else:

print('user has entered',a)

a=input('enter the value of a: ')

if a=='atmiya':

print('user has entered atmiya as input')

print('the value got matched')

else:

print('user has entered',a)

print('the value does not got matched')

a=int(input('enter the value of a: '))

b=int(input('enter the value of b: '))

if a>b:

print('the value of a is smaller than value of a')

else:

print('the value of a is smaller than value of b')

**Dt -0-03-2023**

# if... elif... else

when there is need of testing multiple condition and execute statements depending on those condition you can write several elif statements between if & else

Syntax:

if condition:

stetement1

elif condition:

stetement2

elif condition:

stetement2

else:

stetement3

when condition 1 is True, the statements 1 will be executed.

if condition 1 is false then condition 2 is eveluated.

when condition 2 is True, the statements 2 will be executed. when condition 2 is false, then condition 3 is tested, if condition 3 is True, then statements3 will be executed.when condition 3 is False, the statements 4 will be executed.

it means statements 4(statements under else) will be executed only if none of the conditions are true

**program to know whether the given number is positive, negative or zero**

a=int(input('enter the number: '))

if a>0:

print(a, 'is a positive number')

elif a==0:

print('it is zero')

else:

print(a, 'is a negative number')

a program to print the entered number into word

# else is not menedadery while using if... and elif...

a = int(input('enter the number: '))

if a == 0:

print("zero")

elif a == 1:

print("one")

elif a == 2:

print("two")

elif a == 3:

print("three")

elif a == 4:

print("four")

elif a == 5:

print("five")

else:

print("the range is between 0 and 5")

**discount(0, 5000+10, 8000+12. 10000+15, 200000+25 )**

amount=int(input('please enter the amout: '))

if amount<=0:

print('bill amount must be more than 0')

elif amount<=5000:

print('purchase little more to avail discount')

elif amount>=5000 and amount<=8000:

print('you got 10 percent discount')

elif amount>=8000 and amount<=10000:

print('you got 12 percent discount')

elif amount>=10000 and amount<=20000:

print('you got 15 percent discount')

else:

print("hurry!, you got a maximum discount of 25 percent")

**result**

marks=int(input('enter your marks: '))

if marks<50:

print('you need to re-appear in exam')

elif marks>=50 and marks<=60:

print('you passed the exam')

elif marks>=60 and marks<=70:

print('you passed the exam')

elif marks>=70 and marks<=80:

print('you passed the exam')

elif marks>=80 and marks<=100:

print('you passed the exam')

else:

print('enter valid marks: ')

**discount(0, 5000+10, 8000+12. 10000+15, 200000+25 )**

amount=int(input('please enter the amout: '))

if amount<=0:

print('bill amount must be more than 0')

elif amount<=5000:

print('purchase little more to avail discount')

elif amount>=5000 and amount<=8000:

print('you got 10 percent discount')

discount=(amount\*10)/100

print('amo

elif amount>=8000 and amount<=10000:

print('you got 12 percent discount')

elif amount>=10000 and amount<=20000:

print('you got 15 percent discount')

else:

print("hurry!, you got a maximum discount of 25 percent")

**Dt -270-03-2023**

# **while loop**

a normal statement executed top to bottom only once.

but we can use while loop certain block of statement number of times depending on whether condition becomes true of false

syntax:

while condition:

statements

here statements represent one statement or block of statement

an interpreter first checks the condition if the condition is true, then it will execute the statements return after the colon(:)

after executing the statements it will go again to check condition

if the condition is found true, it will execute the statements

this process will be repeated as many times the condition becomes true

as soon as the condition becomes false, the interpreter will come out of the while loop

a=1

while a<=10:

print(a)

a=a+1

print('end of while')

allowing user to enter numbers, as soon as he insert 0, program will give sum of all the entered numbers

number=int(input('enter a no'))

while number!=0:

total+=number

number=int(input('enter a no'))

print('the sum of entered no', total)

# table of 1

num=int(input('enter the no you what to display table: '))

counter=1

while counter<=10:

ans=counter\*num

counter=counter+1

print(num,"x",counter,"=","ans")

**to display even numbers between 1 and 10**

a=1

while a>=1 and a<=100:

print(a)

# a=a+2

a+=2

**even or odd base on user choice(1 for odd and 2 for even)**

num=int(input('ennter your choice either 1 or 2:'))

if num==1:

print('program will print odd numbers between 1-10')

while num>=1 and num<=10:

print(num)

num=num+2

elif num==2:

print('program will print even numbers between 1-10')

while num>=2 and num<=10:

print(num)

num=num+2

else:

print('enter either 1 o 2')

**program to find and display only even numbers from the given set of numbers**

a=[2,5,3,6,2,5,25,5,11,8654,6,3,59]

b=0

while (b<len(a)):

if a[b]%2==0:

print(a[b])

b=b+1

**accept string from the user and print each character one by one**

a='atmiya'

b=0

while(b<len(a)):

print(a[b])

b=b+1

**Dt -28-03-2023**

# **the for loop**

it can be used to execute a group of statements repeteadly depending upon the number of elements in the sequence

the for loop can work with sequence like string, list, tuple, range etc.

the first elements of sequence is assigned to the variable is written after 'for' and then the statements are executed. next the second element of the sequcne is assigned to the variable and the statements are executed second time.

in this way, for each element of the sequence, the statements are executed.

so the for loop is executed as many times as there are number in the sequence

syntax:

for var sequence:

statements

a='atmiya'

for ch in a:

print(ch)

**display elements of the string using index**

a='atmiya';

n=len(a)

for i in range(n):

print(a[i])

display 0-9

for i in range(10):

print(i)

range start with 0 and end with (n-1)

display odd number between 1-10

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

print(i)

**display even number between 1-10**

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

print(i)

display no from 10 - 1

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

print(i)

**printing each elements from the list one by one**

lst=[12,25,2,55.33,"atmiya", [12,3,633.6], ]

for i in lst:

print(i)

**adding all the elements of the list**

lst=[12, 25, 2, 55.33, 25 -96, 45 ]

sum=0

for i in lst:

sum+=i

print("sum is=",sum)

**using else with for**

lst=[1,2,5,2,5,22,5]

for i in lst:

print(i)

else:

print('all the elements of list are printed')

**finding square of each element of the list**

lst=[1,2,5,2,5,22,5]

abc=0

n=len(lst)

total=0

for i in range(n):

# print(lst[i])

abc=lst[i]\*\*2

total+=abc

print(lst[i],"\t",abc)

print('sum of the square is: ', total)

**Dt -29-03-2023**

# **nested loops**

nested loops mean loop inside loop

in python programming language there are two types of loops 1. while loop 2.for loop

using this loop we can create nested loops

for example while loop inside the for loop, for loop inside the for loop etc.

the inner loop will be executed 1 time for each iteration of the outer loop

structure:

outer loop:

inner loop:

inner loop:

for i in range(2):

for j in range(3):

print("i=",i,"j=",j)

for i in range(0,2):

for j in range(0,3):

print("i=",i,end=" ")

for i in range(2):

for i in range(3):

for i in range(4):

print("i=",i)

for i in range(3):

for j in range(4):

print(i, end=" ")

print()

NOTE: outer loop is incharge of rows and inner loops is incharge of columns

create seqare/rectangle by acceting rows and columns and symbole from user

rows=(int(input("enter number of rows:")))

columns=(int(input("enter number of columns:")))

symbol=(input("enter number of symbols:"))

for i in range(rows):

for j in range(columns):

print(symbol, end=" ")

print()

**left aligned, right aligned triangle**

for i in range(0,rows):

for j in range(0,i):

print(symbol, end=" ")

print()

**Dt -01-04-2023**

# **break and continue**

### **the break statement**

the break statement in python works as a loop control statement

it can be used in side a for loop, while loop or nested loop to come out of the loop

when break statement is executed the python interpreter jump out of the loop to process the next statement

break statement is used to bring the control out of the loop when some external condition is triggered

it terminates the current loop

that is the loop in which is apears and resumes executions at the next statement immeditely after the end of that loop

if the break statement is inside a nested loop, the break will terminate the inner most loop

**search whether the number inserted by user is available in the list or not**

a=[1,2,33,6,9,7]

search=int(input('enter an element to search: '))

for element in a:

if search== element:

print('element is found in the list')

break

else:

print('element is not available in the list')

break;

print 10 to 5

a=10

while a>=1:

print(a)

a-=1

if a==4:

break;

print('out of the loop')

using break with string

a='Atmiya'

for b in a:

print(b)

if b=='i':

break

**same program as above using while**

a='atmiya'

i=0

while True:

while 1:

while 0:

print(a[i])

if a[i]=='i':

break

i+=1

while (i==0):

### **break statement in nested loop**

lst=[1,2,3,4,5]

lst2=[6,7,8,9,10]

for i in lst:

for j in lst2:

print(i, j)

if i==3 and j==8:

if i==7 and j==9:

if i==1 and j==6:

print('break')

break

a='atmiya'

for b in a:

if b=='m':

break

### **the continue statement**

a='atmiya university - rajkot'

for i in a:

if i=='a':

continue

print(i, end="")

print 1-10, except 5 and 6

for i in range(1,11):

if i==5 or i==8:

continue

print(i, end=" ")

**Dt -03-04-2023**

# pass statement in python

pass statement is used as 'placeholder' for future code.

when the pass statement is executed, nothing happens, but you avoid getting an error when empty code is not allowed.

empty code is not allowed in loops, fucntion definitions, class definitions or in if statements(some times).

usin pass inside loop, function or a class gives you a ficility to keep it empty

pass statement is used when we need a statement syntectically but we do not do any operation

the difference between pass and comment is that comment is ignored by the interpreter where is pass is not ignored.

a=0

while a<10:

a=a+1

if a>5:

pass

print(a)

stub(dummy code)

**program to display only negative values from the list**

a=[1,2,3,6,-96,-3,2,-7,14,-36]

for i in a:

if i>0:

pass

else:

print(i)

# **Array in python**

an array is an object that stores a group of elements with same data type.

the main advantage of an array is to store and process a group of elements easily

an array can store only one type of data. it means we can store only integer type or only float type elements in to an array at a times.

but we can not store one integer value, one float and one character type element into the same array.

an array can increase of decrease there size size dynamically. it means we need not declare the size of an array. where element are added it will increase it size and when element are removed it wwill automatically decrease it size in memory.

## **advantage of an array**

arrays are similar to list. the main difference is that array can store only one type of elements list can store different type of elements

when dealing with huge numbers of elements array uses less memory then list

arrays offer faster execution then list

import array

syntax:

arrayname = array(type code, [elements])

### **creating an array**

Table form the screenshot

to deal with array we need to import array module

three ways to imoport array module into our program

when we import array module we are able to get the array class of that module that helps us to create an array

**way-1**

import array

a=array.array('i', [10,206,-90.33])

a=array.array('i', [10,206,-90])

print(a)

here the first 'array' represents a module name and the next 'array' represents the class name for which object is created.

**way-2**

import array as ar

b=a.array('f', [10,206,-90.36,])

print(b)

here 'ar' is an alias of array module

**way-3**

from array import \*

c=array('i', [1,2,5,5,55,95,-96])

print(c)

##### **using chracters in the array by using type code 'u'**

from array import \*

c=array('u', ['a','d','e','t'])

c=array('u', ["a", "b","c"])

print(c)

the meaning of this statement is import all(classes, objects, function, etc) from the array module into our program so there is no need to mention array module name while creating

**Dt -08-04-2023**

# indexing & slicing

### **Indexing**

an index represents the position number of elements in the array. for example when we create a array like this(a=array('i', [10,20,30,40,50,60])). python interpreter allocates 5 blocks of memory

here 0,1,2 extra are representing the position numbers of the elements. so in general we can use "i" to preresent the position of any element.

this "i" is called index of the array

### copying the elements of an array to another array

from array import \*

c=array('u', ['a', 't', 'm', 'i', 'y', 'a'])

d=array(c.typecode, (a for a in c))

for ch in d:

print(ch)

### indexing and slicing on arrays using array index and for loop

a=array('i', [10,20,30,40,50,60])

n=len(a) #finds numbers of elements in an array

print('length of the array: ',n)

for i in a:

print(i, end=" ")

for i in range(n):

print(a[i], end=" ")

### indexing and slicing on arrays using array index and while loop

i=0

while (i<n):

print(a[i], end=" ")

print(i)

i=i+1

## Slicing

a slice represent a peace of the array. slicing is useful to retrive arrange of elements. the general format of a slice is

format:

# array\_name(start : stop : stride/jump)

we can eleminite any one or any two from the above three

c= array('u', ['a','t','m','i','y','a','u','n','i','v','e','r','s','i','t','y'])

a=c[0:15]

a=c[0:16]

a=c[1:16]

a=c[:16]

a=c[0:]

a=c[:4] #0to3 (last index is excluted)

a=c[3:8]

a=c[-16:]

a=c[-16:-5]

a=c[-16:5]

a=c[1:15:2]

print(a)

for i in c[0:12]:

print(i, end=" ")

for i in c[0:12:2]:

print(i, end=" ")

indexing vs sliceing

indexing -> whole array

sliceing -> selective/user wanted elements

**Dt -19-04-2023**

# processing the arrays

# the arrays class of arrays module in python offers methods to process arrays easily.

# the programmers can easily preforms sertain operations by using this methods

# array methods(from screenshots table)

from array import \*

ar= array('i', [1,2,3,4,5,6,])

# print("the initial values inserted in the array: ", ar)

### appending values to the array

# ar.append(7)

# ar.append(8)

# ar.append(9,10) #invalid task(can take exactly one value at a time)

# print("an array value after the appendings: ", ar)

# insert()

ar.insert(0,11)

# ar.insert(4,44) #first position is index no

ar.insert(44,444)

ar.insert(4,4444)

# print("an array value after the insert: ", ar)

# if the position specified is not available, than value will be inserted after the last element in the array

# print(ar[44])

# print(ar[7])

# removing the element

# ar.remove(88)

# ar.remove(3)

# removing the last element

# pe=ar.pop()

# print("the poped element was: ",pe)

# print("elements of array after pop: ",ar)

# finding the position of an element

# a=ar.index(4)

# a=ar.index(14)

# print('the position of an element is: ',a)

# removing as per the index/position

# ar.pop()

# print(ar)

# convert an array into a list

# lst=ar.tolist()

# print(lst)

# print(ar)

# taking mark of student, finding total of marks and percentage

# str1=input("enter makrs: ").split(' ')

str1=input("enter makrs: ").split(',')

marks=[int(num) for num in str1]

print(marks)

total=0

for a in marks:

print(a)

total+=a

print("total marks: ",total)

l=len(marks)

print("total subjects are: ",l)

print("percentage are: ",total/l)

# convert vs copy

# pop() vs remove()

# insert() vs append()

**Dt -20-04-2023**

# Searching an element from the array:

from array import\*

a=array('i',[])

print('How many elements you want to enter?:')

n=int(input())

for i in range(n):

print('Enter an element:')

a.append(int(input()))

print('Elements of an array:',a)

print('Which element you want to search?:'),end='')

srch=int(input())

try:

posi=a.index(srch)

print('Element found at position:',posi+1)

except ValueError:

print('Element not found in the array')

# Types Of ARRAY

## 1)single dimensional array

this array represents only one row or one column of elements.

for example:

Marks obtained by student in 5 subject can be return as

marks=array('i',[20,30,40,50,60])

the ABOUT CONTAIN (MARKS OF ONE STUDENTS)AN ITS CALLED SINGLE DIMENSIONAL ARRAY OR ONE DIMENSIONAL ARRAY

## 2)MULTI DIMENSIONAL ARRAY

this array represent more than one row and more than one column of elements

For example: marks obtained by 3 students in 5 subjects can be written has

marks=array(‘i’,[20,30,40,50,60],

[20,30,40,50,60]

[20,30,40,50,60])

Python Supports only Single dimensional but we can create multi-dimensional using third package like ‘numpy’ (Numeric Python)

Working with arrays using numpy

numpy is package that contain several classes, function, variables,etc.To deal with scientific calculator in python

we can create single dimensional as well as multi dimensional using numpy

numpy contains a large library of mathematical function.

generally array are which created using numpy are called and dimensional array

if numpy module is not found then install it using ‘pip install numpy in cmd’

to work with numpy we must first import numpy modules into our python program.

# numpy

### import numpy

ar= numpy.array([1,2,3,4,5,6]

print(ar)

### import numpy as numpy

ar=np.array([1,2,3,4,5,6])

print(ar)

### from numpy import\*

ar=np.array([1,2,3,10,4,5,6])

print(ar)

from numpy import\*

ar=array(['a','b','c','d','e','f])

print(ar)

from numpy import\*

ar=array(['apple','bannana','2','chocolate'])

print(ar)

## Creating array using linspace

#syntax: linespace(start,stop,n)

#start means: starting element,

#stop means: ending element,

#n means: number of parts elements should be divide (by default 50)

from numpy import\*

a=linespace(2,10,5)

print(a)

### Creating array using logspace

#syntax: logspace(start,stop,n)

#start means: starting element (10^start),

#stop means: ending element (10^end),

#n means: number of parts elements should be divide (by default 50)

from numpy import\*

a=linespace(2,10,5) #10^2......10^10

print(a)

### Creating array using arrange function

#Syntax: arrange(start,stop,stepsize)

from numpy import\*

a=arrange(2,10,2)

print(a)

**Dt -21-04-2023**

# operations on array

from numpy import \*

ar=array([40,20,30,60,90])

print(ar)

# print("adding 5 to each element of array: ", ar+5)

# print("subtractinging 5 to each element of array: ", ar-5)

# print("MULTIPLYING 5 to each element of array: ", ar\*5)

# print("DIVIDING 5 to each element of array: ", ar/5)

# print("the biggest element of array: ", max(ar))

# print("the smallest element of array: ", min(ar))

# print("the addition of all elements of array: ", sum(ar))

# print("the average of all elements of array: ", mean(ar))

## comparing arrays

# from numpy import \*

# a=array([40,20,30,60,90])

# b=array ([40,40,30,90,60])

# print(a==b)

# print(a!=b)

# print(a>b)

# print(a<b)

# print(a>=b)

# print(a<=b)

## compare corresponding elements of two arrays and retrieve the biggest elements

# from numpy import \*

# a=array([40,20,30,60,90])

# b=array ([40,40,30,90,60])

# # c=a>b, a,b

# c=where(a>b, a,b) #use of where() function

# c=where(a<b, a,b)

# c=where(a<b, b,a)

# print(a)

# print(b)

# print(c)

# the where functin can be use to create new array based on whether a given condition is true or false

# the syntax of the where functin

# syntax: array=where(condition, expression1, expression2)

# the use of where() function is demonstrated in the above example

# a program to retrive non zero element from the array

# from numpy import \*

# a=array([4, 0,20,30,6, 0,9, 0, ])

# b=nonzero(a) #use of nonzero()

# print(a)

# print(a[b])

## the nonzero() function

# the nonzero() function is used to retrieve the non zero elements from the array

# program to create a view of an existing array

# from numpy import \*

# a=array([4, 0,20,30,6, 0,9, 0, ])

# b=a.view() # here 'b' array is mirror of array 'a'

# print(a)

# print(b)

# print(id(b))

# print(id(a))

# b[2]=320

# print(a)

# print(b)

# a[3]=-320

# print(a)

# print(b)

# view- mirror effect / mirror creator

# a view() function create same array as existing array

# the original array and the new created array will share different memory locations

# if the newly created array is modified, the original array will also be modified and vice-versa

# view() function creates a copy which is known as 'shallow copying'(chicharu) Vs 'deep copy'

# the use of view() function is demonstrated in the above program

# in case we want both array but modifing one array must not have and effect on another array the we may use the copy function.

# the use of copy() function is demonstrated in the below program

# for deep copy in copy() function is use

# from numpy import \*

# a=array([4, 0,20,30,6, 0,9, 0, ])

# b=a.copy() # here 'b' array is mirror of array 'a'

# print(a)

# print(b)

# print(id(b))

# print(id(a))

# b[2]=320

# print(a)

# print(b)

# a[3]=-320

# print(a)

# print(b)

**Dt -26-04-2023**

# slicing and indexing numpy arrays

# slicing refers to extracting a ragne elements of array

# the format of slicing operation is as below

# arrayname[start: stop: stepsize]

# here the default value of 'start' is 0, for 'stop' is n(number of elements), stepsize is 1

# counting starts from 0th position

from numpy import \*

# a=array([1,2,3,4,5,6,7,8,9,10,11])

# print(a)

# retrieving 1st element to 6th elements

# print(a[1:5])

# retrieving 1st element to 6th elements alternatively

# print(a[1:5:2])

# retrieving all the elements

# print(a[:])

# print(a[::])

# retrieving all the elements from 3rd element

# print(a[3:])

# retrieving all the elements from 3rd element alternatively

# print(a[3::2])

# indexing

# indexing refers to location of the elements. by specifying the location of the number from 0 and onwards till n-1, we can refers all elements as a[i] where 'i' can change from 0 to n-1

# i=0

# while(i<len(a)):

# print(a[i], end=" ")

# i=i+1

# attributes of array

# ndim attribute

## # shape attribute

# size attribute

# dtype attribute

# reshape() method

# flatten() method

## the ndim attribute

# the ndim attribute represents the number of dimensions of the array.

# for a single dimensional array it will display 1 and for tow dimensional array it will display 2

# n-number

# dim-dimensions

# a=array([1,2,3,4,5,6,7,8,9,10,11])

# print(a)

# print(a.ndim)

# b=array([[1,2,3,4],[1,2,3,4]])

# print(b)

# print(b.ndim)

# c=array([[[1,2,3,4],[1,2,3,4],[1,2,3,4]]])

# print(c)

# print(c.ndim)

# d=array([[[[1,2,3,4],[1,2,3,4],[1,2,3,4],[1,2,3,4]]]])

# print(d)

# print(d.ndim)

## the shape attribute

# the shape attribute gives shape of array

# the shape of a tuple listing the numbers of elements along each dimensions

# a=array([1,2,3,4,5,6,7,8,9,10,11])

# print(a)

# print(a.shape) #(cols)

# b=array([[1,2,3,4],[1,2,3,4],[1,2,3,4]])

# # b=array([[1,2,3,4],[1,2,3,4],[1,2,3]])#this is not valid

# print(b)

# print(b.shape) #(row, cols)

# b.shape=(4,3) #(row, cols)

# b.shape=(4,2) #(row, cols) #this is not valid

# b.shape=(6,2) #(row, cols)

# print(b)

# print(b.shape) #(row, cols)

## the size attribute

# the size attribute gives the total number of elements in the array

# a=array([1,2,3,4,5,6,7,8,9,10,11])

# print(a.size)

# b=array([[1,2,3,4],[1,2,3,4],[1,2,3,4]])

# print(b)

# print(b.size)

# d=array([[[[1,2,3,4],[1,2,3,4],[1,2,3,4],[1,2,3,4]]]])

# print(d)

# print(d.size)

## the dtype attribute

# the dtype attribute gives the datatype of the array elements

# a=array([[1,2,3,4],[1,2,3,4]])

# print(a.dtype)

# a=array([[1,2,3,4],[1,2,3,4.4]])

# print(a.dtype)

# a=array([[1,2,3,4],[1,2,3,'a']])

# print(a.dtype)

## the reshape() method

# the reshape() method is used to change the shape of an array

# a=array([1,2,3,4,5,6,7,8,9,10])

# # a=array([1,2,3,4,5,6,7,8,9]) #this is not valid

# a=a.reshape(2,5) #(row, cols)

# print(a)

## the flatten() method

# the flatten() method is usefult to collapse the elements of an array into one dimensional array

# a=array([[1,2,3,4],[1,2,3,'abc']])

# a=a.flatten()

# print(a)

**Dt -28-04-2023**

# multi dimensional arrays

# the 2d arrays, 3d arrays etc. are hold multidi-mensional arrays

from numpy import \*

# a=array([[10,11,12,13],[14,15,16,17]])

# print(a)

# the internal memory allocted to all the elements would be in a single row containing 8 digit blocks(at there 8 elements in the array)

# the elements are stored in the contiguous memory location as shown below

# (screenshot)

## # the reshape() function

# this function is used to convert a 1d array into a multi-dimensional array

# it would written as below

# reshape(arrayname, (n,r,c)) here n is optional argument

# here 'n' represents the number of array in the output, 'r' represents row and 'c' represents number of columns

# a=array([1,2,3,4,5,6,7,8,9,10,11,12])

# print(a)

# b=reshape(a, (4,3))

# b=reshape(a, (3,4))

# b=reshape(a, (3,3,2))

# print(b)

## indexing in multi-dimensional array

# index represents the location number.

# the individual elements of 2d array can be accessed by specifying the location no of the row and columns of the element in the array as

# a[0][0] #it represents the 0th row and 0th column

# a[1][3] #it represents the 1st row and 3rd column

# a=array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])

# print(a)

# for i in range(0,len(a)): #for rows

# # print(a[i])

# # for j in range(0,len(a[i])): #for columns

# # print(a[i][j], end=" ")

# # print(a[i],[j])

## slicing in multi-dimensional arrays

a=array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])

print(a)

# print(a[0,:]) #(n,c)

# print(a[2,:])

# print(a[:,0])

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

# print(a[2:3,1:2])

# print(a[2,:1])

# print(a[2,:2])

# print(a[0,1])

# print(a[0,0:4])

# print(a[0,1:3])

# print(a[0:1])

# print(a[0,2: 1])

# print(a[2: ])

print(a[0,0])

print(a[0,1])

print(a[0,2])

**Dt -29-04-2023**

# s1='Welcome to Atmiya University, Rajkot' #Both s1 and s2 will have same output

# s2="Welcome to Atmiya University, Rajkot" #Both s1 and s2 will have same output

# s3='Welcome to "Atmiya University", Rajkot' #To have quote in the output

# s4='Welcome to \t Atmiya University, \nRajkot' #To give tab and print on new line (escape character)

# s5=r'Welcome to \t Atmiya University, \nRajkot' #r is used to nulify the effect of escaper character

# print(s1)

# print(s2)

# print(s3)

# print(s4)

# print(s5)

# s6=u'\u7770' #Unicode

# print(s6)

# print(len(s1)) #To find no of characters in the string

# s7="atmiya"

# s8="University"

# s=s7+s8 #String Concatenation

# print(s)

#A program to access each element of a string in forward and reverse order (using While)

# str='Atmiya University'

# n=len(str)

# i=0

# while i<n:

# print(str[1], end=' ')

# i=i+1

# print()

#In the reverse order

# n=len(str)

# while i<=n:

# print(str[-i], end =' ')

# i=i+1

# #A program to access each element of a string in forward and reverse order (using for loop)

# str='Atmiya University'

# i=0

# for i in str:

# print(i, end=' ')

# #Reverse

# for i in str[::-1]:

# print(i, end='|')

# #Slicing

# str='Atmiya'

# print(str[0:6:1]) #[:::]

# print(str[0:6:2])

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

#Checking whether string exist in the main string

# str=input('Enter the string: ')

# substr=input('Enter the seach string: ')

# if substr in str:

# print(substr, 'is availabe in the main string')

# else:

# print(substr, 'is not available in the main string')

**Dt: 01-05-2023**

**counting sub-string in string**

# format: stringname.count(substring, beg, end)

# finds in given range

# str="new india new dreams"

# n=str.count('new', 0,20)

# n=str.count('new')

# n=str.count('new', 5 , 10)

# print(n)

# replacing a string

# str="good morning"

# s1="morning"

# s2="evening"

# str=str.replace(s1, s2)

# print(str)

# changing case of a string

# str="Your Future is Bright"

# print(str.upper())

# print(str.lower())

# print(str.swapcase())

# print(str.title())

**# string testing methods**

# (screenshot)

# mobile\_number=input("Enter Mobile Number: ")

# print(mobile\_number.isdigit())

# if mobile\_number.isdigit()!=True:

# print("enter valid mobile number")

# sort a string into alphabet order

# str=[]

# n=int(input("how many strings you want to enter: "))

# for i in range(n):

# print("enter the string: ", end="")

# str.append(input()) #use of append method

# print(str)

# str.sort() #use of sort method

# print(str)

# for i in str:

# print(i)

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

# program to search the strings from the

group of strings and also find its position

# str=[]

# n=int(input("how many strings you want to enter: "))

# for i in range(n):

# print("enter the string: ", end="")

# str.append(input()) #use of append method

# print(str)

# ask user for the string to search

# srch=input("Enter the string to search:")

# searching

# for i in range(len(str)):

# if srch==str[i]:

# print("the search string found at the position: ", i)

# break

# else:

# print("the searched string not found")

# flag=False

# searching

# for i in range(len(str)):

# if srch==str[i]:

# print("the search string found at the position: ", i)

# flag=True

# if not flag:

# print("the searched string not found")

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

# working with characters

# str="atmiya"

# # ch=str[0]

# # ch=str[4]

# ch=str[0:3]

# print(ch)

# program to know the the type of character

str=input("enter a character: ")

ch=str[0]

if ch.isalpha():

print("it is an alphabet")

if ch.isupper():

print(" alphabet is upper case")

else:

print(" alphabet is lower case")

else:

print("it is number")

**Dt: 02-05-2023**

**#Unit 3: List, Tuple, Dictionary**

## **List**

#Program to create list using range function #Format of range: range(start, stop, stepsize)

#0-7

# lst1=range (8)#(0:8:1)

# for i in lst1:

# print(i, end=' ')

# print()

# lst2=range(8,16)

# for i in lst2:

# print(i, end=' ')

#odd numbers between 1-10

# print()

# lst3=range (1,10,2)

# for i in lst3: print(i, end=' ')

# print()

#Updating the list

lst4=list(range(1,8))

# print(lst4)

# #Append

# lst4.append(11)

# print(lst4)

# #Update

# lst4[3]=50

# print(lst4)

#Delete by index

# del lst4[3]

# print(lst4)

#Delete by value

# lst4.remove(5)

# print(lst4)

#Finding index of element

# a=lst4.index(2)

# print(a)

#Length of a list

# n=len(lst4)

# print(n)

#Clear the list

# lst4.clear()

# print(lst4)

#Display elements of list in a reverse order

# lst=['sunday', 'monday', 'tuesday']

# a=lst.reverse() #Use of reverse()

# print(a)

# print(lst)

#Concatenation of lists

# a= [1,2,3]

# b=['one', 'two', 'three']

# print(a+b)

#Repetition of list'

# print(a\*3)

**Membership in Lists**

# #Using 'in' and 'not in'

# lst=[1,2,3,4,5,6,7]

# a=7

# b=8

# print(a in lst)

# print(b in lst)

# print(a not in lst)

# print(b not in lst)

#Alliasing a list #copy the elements, modification in any list will be done in other list too.

# lst1=[1,2,3,4,5,6,7]

# lst2=lst1 #Aliasing

# print(lst1)

# print(lst2)

# lst2.append(8)

# print(lst1)

# print(lst2)

#Cloning a list # copy the elements, modification in any list will not effect other list

# lst3=[1,2,3,4,5,6,7]

# lst4=lst3[:]#Cloning

# print(lst3)

# print(lst4)

# lst4.append(8)

#Counting how many times an element is found in the list

# a=[]

# n=int(input('How many elements to insert?: '))

# for i in range(n):

# print('Enter the element: ',end='')

# a.append(int(input()))

# print('The list is: ',a)

# find=int(input('Enter an element to count: '))

# cnt=0

# for i in a:

# if(find==i):

# cnt=cnt+1

# print('{0} is found {1} times.'.format(find, cnt))

#Finding common elements in two lists

# flower1=['rose', 'marigold', 'mogra']

# flower2=['rose','orchid', 'daisy']

#Converting to set

# s1=set(flower1)

# s2=set(flower2)

# s3=s1.intersection(s2)

# print(s3)

#Converting result into list

# common=list (s3)

# print(common)

#Create nested list

lst=[1,2,3, [10,20]]

print('Elements of list:', lst)

print('The first element of list: ', lst[0])

print('The first element of list: ', lst[3])

for i in lst[0:4]:

print(i)

for i in lst[3]:

print(i)

#Create a list ehich contains square of 1-10

sqr=[] #Empty list

for i in range(1,11):

sqr.append

**Dt: 03-05-2023**

**nested tuples**

# tpl=(10,20,30,(40,50))

# print (tpl)

# print (tpl[3])

# taking details of employees in tuple and sort it

# emp=((2,"abc", 3000),(1,"jk", 20000),(3,"dk", 1000))

# print (emp)

# print (emp[2])

# print (sorted(emp))

# sorting on the basis of employee names

# print (sorted(emp,key=lambda emp: emp[1]))

# sorting on the basis of employee salary

# print (sorted(emp,key=lambda e: e[2]))

# print (sorted(emp,key=lambda e: [2]))

# -------------------------------------------------------

# inserting elements in a tuple

# names=('a','b','d','e','f')

# print (names)

# enter new name to insert and position at which to be entered

# lst=[input("Enter new value: ")]

# new\_nm=tuple(lst)

# print (new\_nm)

# pos=int(input('enter the position: '))

# copy names from 0th position to pos-1

# temp\_name=names[0: pos-1] #[start: stop]

# print (temp\_name)

# temp\_name=temp\_name+new\_nm

# print (temp\_name)

# concatenate remaining elements of names from pos-1 till end

# names=temp\_name+names[pos-1: ]

# print (names)

# ------------------------------------------------------

# modify elements in the existing tuple

# insert vs modify

# names=('a','b','d','e','f')

# print (names)

# enter new name to insert and position at which to be entered

# lst=[input("Enter new value: ")]

# new\_nm=tuple(lst)

# print (new\_nm)

# pos=int(input('enter the position: '))

# copy names from 0th position to pos-1

# temp\_name=names[0: pos-1] #[start: stop]

# print (temp\_name)

# temp\_name=temp\_name+new\_nm

# print (temp\_name)

# concatenate remaining elements of names from pos till end

# names=temp\_name+names[pos: ]

# print (names)

# -------------------------------------------------------

# delete an elements from a particular position

num=(10,20,30,40,50,60,70,80)

print(num)

# accept position number from which element need to be deleted

pos=int(input(' enter position number: '))

# copy from 0th position to pos-1 into another tuple

num1=num[0:pos-1]

num=num1+num[pos:]

print(num)

## **dictionary**

# a dictionary represents a group of elements arrange in the form of key-values pairs

# in the dictionary, first element is considered as 'key' and imdiate next is considered as its 'value'

# we can not use slicing or indexing to retrive elements to dictionary

# dt={ 1: "abc", 2: 'pqr', 3: "xyz" }

# print(dt)

# operations on dictionaries:

# dt={ 'name': "chagan", 'id': "101", 'salary': 15000 }

# print(dt)

# print('the id of employees is: ', dt['id']) #retriving values using key

# print('the salary of employees is: ', dt['salary']) #retrieving values using key

# n=len(dt) #returns no of key value pair

# dt['salary']=52000 #updating the value using key

# dt['dept']="MCA" #adding new key: value

# del dt['id'] #deleting a pair of key:value

# print(dt)

# print('id' in dt)

# print('dept' in dt)

# print('id' not in dt)

# print('dept' not in dt)

# dictionary methods

# dt={'name': "chagan", 'id': "101", 'salary': 15000 } #creating dictionary

# print(dt)

# dict=dt.copy() #copys the content of dt to dict

# print('the values of dict:', dict)

# a=dict.keys() #to retrieve all the keys from the dictionary

# a=dict.values() #to retrieve all the values from the dictionary

# print(dict.get('name')) #displaying the value of specified key

# print(dict.get('id'))

# print(dict.items()) #returns a list containing a tuple for each key value pair

# print(dict.pop('id')) #pops the value associated with key

# print(dict)

# print(dict.clear()) #removes the all the keys and values, means empty the dictionary

# print(dict)

# print(type(dict))

# program to take subject and makrs from student and display total marks, display the marks of entered subject

# marks={}

# print('how many subjects you want to enter: ', end="")

# n=int(input(''))

# for i in range(n):

# print('enter subject name: ')

# key=input()

# print('enter mark: ')

# val=int(input())

# marks.update({key:val})

# print(marks)

# total=sum(marks.values())

# print('total marks are: ',total)

# print('enter name of subject to know it marks: ')

# sub=input('')

# mar=marks.get(sub,-1)

# if mar==-1:

# print('invalid subject')

# else:

# print('the makrs in {} are {}'.format(sub, mar))

# mar=marks.get(sub)

# if mar==None:

# print('invalid subject')

# else:

# print('the makrs in {} are {}'.format(sub, mar))

# sorting the elements of a dictionary using lambda function

# a lambda is function that does not have a name and can be written without 'def' keyword

# they are used to perform some calculation or processing easily

# a=lambda x,y: x+y

# print(a(10,20))

# there are two arguments in the function named x & y, after colon(:) we wrote the body i.e.(that is) x+y, and this is the value returned by the lambda function

colors={9:'blue', 7:'red', 2:'green', 4:'yellow'}

c1=sorted(colors.items(), key=lambda a:a[0]) #sorting on the basis of the key using lambda function

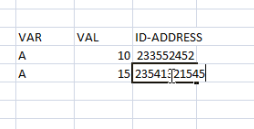
print(c1)

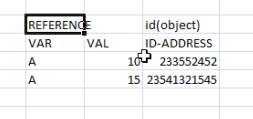
c1=sorted(colors.items(), key=lambda a:a[1]) #sorting on the basis of the key using lambda function

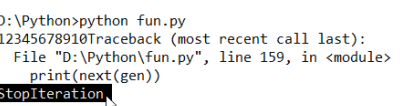
print(c1)

**Dt: 04-05-2023**

**Dt: 05-05-2023**

****

****

****

**returning multiple values from a function**

# a function in python can ruturn multiple values

# when function calculate multiple results and wants to return all those results we can use return statement as "return a,b,c"

# this values are writted by a function in the form of tuple

# performing basic arithmatic operations using functions

# def arith(a,b):

# add=a+b

# sub=a-b

# mul=a\*b

# div=a/b

# return add, sub, mul, div

# p,q,r,s=arith(10,50)

# print("the result is: ", p,q,r,s)

### pass by object reference / pass by object

# immutable objects in python: int, float, string, tuple,

# mutable objects in python: list

# using: immutable mutable object int

# def modify(a):

# a=16

# print('the value of a inside the function is: ', a,id(a))

# #call the function

# a=10

# modify(a)

# print('the value of a outside the function is: ', a, id(a))

# using: mutable object list

# def modify(lst):

# lst.append(15)

# print(lst, id(lst))

# # calling the function

# lst=[10,20,30,40,50,60]

# modify(lst)

# print(lst, id(lst))

# as the value assigned to variable 'a' is of integer type, modifying the value of 'a' will be not be stored at same memory address as integer is immutable, modifying value of a will occupy another memory to store a new value of new 'a'.

# formal and actual arguments

# def sum(a,b): --> fromal arguments

# a=10,b=20

# sum(a,b) --> actual arguments

# the actual arguments are of 4 types:

# - positional arguments

# - keyword arguments

# - default arguemnts

# - variable length arguments

# positional arguments

# what is position arguments?

**from ast import arguments**

# the number of arguments and their position in the function definition should match exatly match and positon of the arguments in the function call

def combi(s1,s2):

s3=s1+s2

print(s3)

# calling the function

combi("good", "morning") #i/O: good morning

combi( "morning", "good" ) #i/O: morning good

combi( "morning", "good", "student" ) # take 2 positional arguments but 3 were given

## **keyword arguments**

# keyword arguments are arguments that identify the paramenters by their names

# at the time of calling function, we have to pass two values and we can mansion which value is for what

# def student (rno, name):

# print("roll is: ", rno)

# print("name is: ", name)

# student(name='abc', rno=55)

# student( rno=15, name='dk')

# default arguments

# if required we can set a default value to the parameter if at the time of calling function the value is not passed to the argument then default value will be used. and if the value is passed to the argument then the passed value wil be used

# def stud(no, name='user'):

# print("roll no is: ", no)

# print('name is: ', name)

# stud(10)

# stud(10, 'jk')

# variable length arguments

# in some situation it is possible that the programmer is unware about the requirements of the paramenters in the program if in program two parameters are declared, and while using it user feel to give more then two values then error will occur.

# in that case variable length argument can be used.

# a variable length argument is an argument that can accept any number of values

# formet: def name\_of\_function (farg, \*args)

# def add(farg, \*args):

# sum=0

# for i in args:

# sum=sum+i

# print("sum of all numbers: ", (farg+sum))

# add(10,20)

# add(10,20,20)

# add(10,20,30,44.44)

# passing a group of elements to a function

# some time it is required to receive group of elements like numbers or strings, we can accept them in to a list and then pass the list to the function

# example:

# def show(lst):

# for i in lst:

# print(i)

# # taking group of strings as an input

# print("enter strings seperated by space: ")

# lst=[a for a in input().split(' ')]

# show(lst)

# anonymous function(which is also known as lambda)

# we already know that while writing a function we need to give name to the function

# we usually do it like

# def name\_of\_function(argument)

# but the anonymous function it is definded with the keyword lambda

# def square(a):

# return a\*a

# this could be written in anonymous function as:

# lambda a:a\*a

# find the square of the entered number

# a=int(input(' enter the number: '))

# ans=lambda a:a\*a

# print("the square of the entered number is: ", ans(a))

# big=lambda a,b:a if a>b else b

# a,b=[int(n) for n in input('enter two numbers comma seperated: ').split(',')]

# print(f'the bigger number out of two number is : ', big(a,b))

# generators

# generators are function that return a sequence of values. a generator function is written like ordionary functions but it uses 'yield' statement.

# the 'yield' statement is usful to return the value.

# what is meant by yield statemnt? it is used in generators

# program to create generator that returns a sequence of number from x to y

def gen1(x,y):

while x<=y:

yield x

x=x+1

# fill generator object with 1 and 10

gen=gen1(1,10)

# display the values

for i in gen:

print(i, end="")

print(next(gen))

# function vs generator

**Dt: 09-05-2023**

**unit -4**

# errors in a python

# Exception

# an exception is an event which occurs during the execution of the program that changes is the normal flow of the program's instructions

# an exception is runtime error which can be handled by the programmer

# it means if the programmer can gus an error in the program end he can do something to eleminate the harm caused by the error, then it is called an exception.

# if the programmer can not do anything in case of error, then it is called an error not an exception

# all exceptions are represented in classes in python. the exception which are already available in python are called built-in exception

# the base class of all built-in exception is 'BaseException' class

# Exception Handling

# exception handling is process of responding to unwanted or unexpected events when a computer programs runs

# exception handling deals with this events to avoid to avoid the program or system crashing

# the purpos of exception handling is to give a proper error message to the user when an error occurs

# to handle exception program should perform following three steps:

# step:1 - the programmer shoudl observe the statement in his program where there may be possibility of exception. such statements must be written in side the 'try' block

# syntax:

# try:

# statements

# if an error aries inside the 'try' block the program will not be terminated.

# the PVM will understand that there is an exception, it will jump into an 'except' block

# step:2 - the programmer shoudl write the except block where he should display the exception details to the user.

# the programmer should also display an error message regarding what can be done to avoid this error

# syntax:

# except exceptionName:

# statements

# statements written inside an 'except' block are called 'handlers' since they handle the situation

# step:3 - the programmer should perform clean up actions like closing the file and terminating any other process which are running. it should be written 'finally' block

# syntax:

# finally:

# statements

# the speciality of 'finally' block is that the statements inside the finally block are executed irrespective of whether there is an error or not

# performing the above the task is called excption handling

# Note: - a single 'try' block can be followed by multiple 'except' blocks

# - 'except' blocks can not be written with try block

# - writing 'except' block and finally block is not mandatory

# - 'finally' block if written then will always executed

# error vs exception

# a=10

# if a==10:

# print('the value of a is 10')

# def contact(a,b):

# try:

# print(a+b)

# except:

# print('cannot concat string with integer')

# contact("i like mango", 100)

# fruits=['mango', 'banana', 'apple']

# try:

# print(fruits[3])

# except:

# print('element does not exist')

# example

# try:

# a=int(input("enter a value: "))

# b=int(input("enter a value: "))

# print(a/b)

# except:

# print('number cannot be divided with 0')

# def increment(sal):

# sal=sal+sal\*15/100

# return sal

# print(increment(4000))

# try:

# print('you are in the try block')

# a=int(input("enter a value: "))

# b=int(input("enter a value: "))

# c=a/b

# except:

# print('number cannot be divided with 0')

# else:

# print('you are in else part')

# print('the answer of division is: ', c)

# finally:

# print('you are in finally block')

**Dt: 10-05-2023**

# try:

# print("you are in the try block")

# a=int(input("enter a value:"))

# b=int(input("enter a value:"))

# c=a/b

# except:

# print("you are in except block")

# print("number cannot be divide with 0")

# else:

# print("you are in else block")

# print("the answer of division is ", c)

# finally:

# print("you are in finally block")

# type of exception (built in exception)

**handling multiple exceptions in one try**

# def avg(list):

# total=0

# for i in list:

# total=total+i

# average = total/len(list)

# return average, total

# call the function

# try:

# tot, avg =avg([1,2,3,4,5,6,7,8,9,'a'])

# except TypeError:

# print("you are not in the try block")

# except ZeroDivisionError:

# print("you are not in the try block")

# handling multiple exceptions in one try

# def avg(list):

# total=0

# for i in list:

# total=total+i

# average = total/len(list)

# return average, total

# # call the function

# try:

# tot, avg =avg([1,2,3,4,5,6,7,8,9,0,'a'])

# except (TypeError,ZeroDivisionError):

# print("please provide number only")

# try:

# tot, avg =avg([1,2,3,4,5,6,7,8,9,0,'a'])

# except (TypeError,ZeroDivisionError):

# if TypeError:

# print("please provide number only:)")

# elif ZeroDivisionError:

# print("please provide number greatern then 0")

# **the assert**

# the assert statement is useful to ensure that given condition is true if it is not true it raises an

# AssertionError

# syntax:

# assert condition message[message not mandatory]

# assertion method can be written in two ways:

# 1. if the condition is false then the exception by the name AssertionError is raised along with the message written in the assert statement

# 2. if the error message is not returned then the AssertionError without the message will be displayed

**handle assertion errors**

# assertion without an error message

# try:

# a=int(input("enter a number between a and 10: "))

# assert a>=1 and a<=10

# print("the entered number is: ",a)

# except:

# print("the number is not between 1 and 10")

**assertion without an error message**

# try:

# a=int(input("enter a number between a and 10: "))

# assert a>=1 and a<=10, "the number is not between 1 and 10"

# print("the entered number is: ",a)

# except AssertionError as obj:

# print(obj)

# user defined exception(custom exception)

# version-1

# def validate(name):

# if len(name) <10:

# raise ValueError

# version-2

# def validate(name):

# if len(name) <10:

# raise ValueError('name is shorter than required')

# version-3

# class NameShorter(ValueError):

# pass

# def validate(name):

# if len(name) <10:

# # raise NameShorter

# raise NameShorter('name is shorter than required')

# version-4

# def validate(name):

# try:

# if len(name) <10:

# # raise NameShorter

# raise NameShorter('error')

# except(NameShorter):

# print('name is shorter than required')

# callint the function

# validate("abc")

# final version

# class BirthYear(OSError):

# pass

# year=int(input("enetr your birth year: "))

# your\_age=2023-year

# print("your age is: ",your\_age)

# try:

# if your\_age<=30 and your\_age>20:

# print('you can apply for the job')

# else:

# raise BirthYear

# except BirthYear:

# print('your age is not as required')

# **file handling**

### types of files

# in python there are two types of files

# 1. text files

# 2. binary files

# text files store the data in the form of the characters. For example if we store student name 'abc', it will be stored as three characters and it score is 87 it store it as two characters.normally text file are use to store character or string

# binary files store entire data in the form of bytes that is i.e. a group of 8 bits each. when the data is retrived from the binary file, the programmer can retrieve the data as bytes. binary files can be used to store text, images, audio and video files

### **opening & closing file**

# we should use open() function to open a file. this function accepts 'filename' and open mode in which to open the file

# syntax:

# file handler = open('filename', 'open mode')

### file opening mode

# 1. -w (to write data into a file. if any data is already present in the file, it would be deleted and the present data will be stored)

# 2. -r (to read data from the file. the file pointer is placed at the beginning of the file )

# 3. -a (to append data to the file. appending at the end of the existing data. file pointer placed at the end of the file. if the file is not present it will be created)

# 4. -x (to open the file in the exclusive creation mode. the file creation is fails if the file already exists)

# 5. + (+with 'w','r','a' will allow both reading and writing)

# diff -r(starting position) vs -a(end of the content)

# closing the file

# a file which is opened should be closed using the close() method. if the file is not closed then data may get currept or be may loss data.

# alos if file not closed then it may not free the memory till the file is open

# we can close file using f.close()

# program to create a text file and store data

# creating a file to store characters

# open the file for writing data

f=open("mytest.txt", 'w')

print(type(f))

# accept text from the user

str=input('enter the test: ')

#writing string into the file

f.write(str)

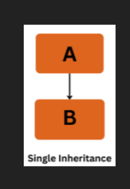
p=f

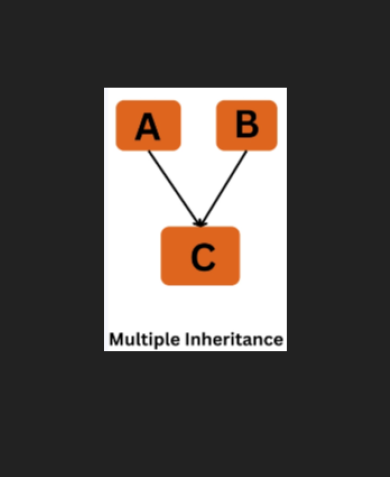
print(type(p))

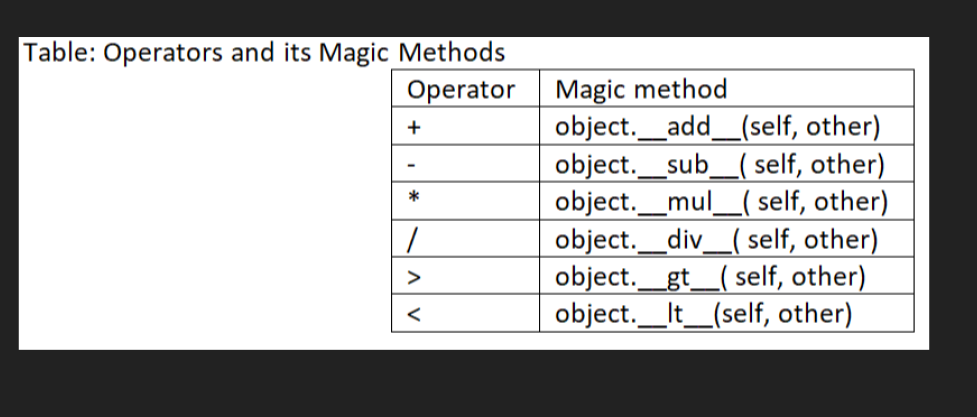
# closing the file

f.close()

**Dt: -29-05-2023**

****

****

****

# class employee:

# def \_\_init\_\_(self, name, degree):

# self.name = name

# self.degree = degree

# def printEmployee(self):

# print('the name of employee is: ' + self.name)

# print('the degree an employee hold is: ' + self.degree)

# e=employee('abc', 'bca')

# e.printEmployee()

# class department():

# def \_\_init\_\_(self, experience, name, degree):

# super().\_\_init\_\_(name, degree)

# self.dept= dept

# self.experience = experience

# def prentdept(self):

# print('the department of employees is: ' + self.dept)

# print('the experience of an employee is: ' + self.experience)

# d=department(10,'MCA)', 'phd')

# d.prentdept()

# d.printEmployee()

## **types of inheritence**

# the main advantage of inheritance is code reusability

# the members of he super/parent class are resuable in the sub/child classes

# there are mainly two types of inheritance

# single inheritance

# multiple inheritece

### **single inheritance**

# a single inheritance enables a derived class to inherit child class to inheritance properties of base/parent class thus enabling code resuablity and the addition of new features to existing code

# class father:

# def func1(self):

# print("father class: function")

# class child(father):

# def func2(self):

# super

# print("child class: function")

# c=child()

# c.func1()

# c.func2()

### **multiple inheritance**

# deriving sub-classes from multiple base-classes is called multiple inheritance

# in this type of inheritence, where will be more then one super/parent classes and there may be one or more sub/child classes

# class father:

# def func1(self):

# print("father class: function")

# class mother:

# def func2(self):

# print("mother class: function")

# class son(father, mother):

# def func3(self):

# print("mother + father=child: function")

# s=son()

# s.func1()

# s.func2()

# s.func3()

# print(son.mro())

# method resulution order(mro)

# explain more with example

# in multiple inheritance sinarios, any specified attribute or method is searched first in the current class if not found, the search continues into parent class in depth-first, left-to-right fashion without searching the same class twice searching in this way is called method resulution order(mro)

# MRO folloes three principals seach for the sub class before going to its base class

# when a class is inherited form several classes, it searches in the order from left to right and last it will not visit any class more then once

# class A:

# def check(self):

# print('in class A')

# class B:

# def check(self):

# print('in class B')

# # classes ordering

# class C(A,B):

# def \_\_init\_\_(self):

# print('constructor C')

# obj=C()

# print(C.mro()) # it prints the lookup order

# # here, C is the name of the class

## Polymorphism

# the word "polymorphism" means having many forms

# in programming polymorphism means he same function name (but diffrent signatures) being use for different types

# the key difference is that the datatypes and number of arguments uses in the function to uderstand lets take an example of wheat flur

# using wheat flur we can make roti, breds, paratha

# the wheat flur is same but we can use it different forms.

# if a vairalbe, object, or method demonstrate different behavior in differnet context, is called polymorphism

**following topics are examples of polymorphism in python**

* duck typing polymorphy
* operator overloading
* method overloading
* mothod overriding

# the datatype of variables does not need to explicitly(manually) declare

# the type is implicitly(automatically) depending upon the purpose for which the variable is used

# a='atmiya'

# print(a)

# print(type(a))

# a=7

# print(a)

# print(type(a))

# class Duck:

# def speak(self):

# print("quack quack ")

# class Human:

# def speak(self):

# print("namaste ")

# def call\_speck(obj): #this method accepts an object and calls speck method

# obj.speak()

# # calling the call\_speak() method and pass and object

# # depending upon the type of object, talk() method is executed

# a=Duck()

# call\_speck(a)

# a=Human()

# call\_speck(a)

### **operator overloading**

# an operators like +, -, \*, /,etc is a symbol that perform some actions as we know that '+' is an operator perform addition when used on numbers

# when '+' is used on strings then strings are concatenated

# when an operator performs different actions it is said to be operator overloading

# operator overloaing

# a=10

# b=6

# print(a+b) #using '+' to add two numbers

# str1='atmiya'

# str2='uni'

# print(str1+str2) #using '+' to concatenate two strings

# lst1=[1,2,3]

# lst2=[11,22,33]

# print(lst1+lst2)

**program to overload greater than(>) operator**

# class ramayan:

# def \_\_init\_\_(self,no\_of\_sloks):

# self.no\_of\_sloks = no\_of\_sloks

# def \_\_gt\_\_(self,other):

# return self.no\_of\_sloks>other.no\_of\_sloks

# class mahabhart:

# def \_\_init\_\_(self,no\_of\_sloks):

# self.no\_of\_slocks=no\_of\_sloks

# r=ramayan(24000)

# m=mahabhart(4000)

# if r>m:

# print('ramayan has more slocks')

# else:

# print('mahabharat has more slocks')

### method overloading

# if a method is written such that it can perform more then one task, it is called method overloading.

# method overloading is not available in python

# in python we can archieve method overloading by writing same method with saveral parameters.

# the method performs the operation depending on the number of arguments in the method call

# class add:

# def sum(self,a=None, b=None, c=None):

# if a!=None and b!=None and c!=None:

# print('the addition of 3 numbers is: ',a+b+c)

# elif a!=None and b!=None:

# print('the addition of 2 numbers is: ',a+b)

# else:

# print('enter atleast 2 numbers')

# a=add()

# a.sum(10)

# a.sum(10,20)

# a.sum(10,20,30)

### method overridding

# when there is method is the parent class, writing the method in the child class so that it replaces the parent class method is called method overriding

# the programmer overrides the parent/super class method when he doen not want to use them in sub/child class

# instead he wants an new functionality to the same method in the sub class

**program to find area of square and circle**

import math

class square:

def area(self, a):

print('the area of square is: ',a\*a)

class circle(square):

def area(self, a):

print('the area of circle is: ',math.pi\*a\*a)

# pi=3.14

c=circle()

c.area(10)

s=square()

s.area(15)

**Dt: -05-2023**