**PYTHON**

Python is a widely used general purpose , high level programming language. It was initially designed by **Guido Van Rossum** in 1991 and developed by Python software foundation . It was mainly developed for emphasis on code readability , and its syntax allows programmers to express concepts in fewer lines of code.

Python is a programming language that lets you work quickly and integrate systems more efficiently.

Why to use python:

1. Python is object oriented: Structure supports such concepts as polymorphism operation overloading and multiple inheritance.
2. Indentation: Indentation is one of the greatest feature in python.
3. It’s open source.( free source)
4. Its powerful

Dynamic typing

Built in types and tools

Library utilities

Third party utilities (eg., Numeric Numpy,…)

1. Its Portable

Python runs virtually every major platform used today

1. Its easy to use and learn.
2. No type Declaration
3. Interpreted language
4. No compilation and linking.

10.Case sensitive. apple Apple name NamE

Applications:

1. Web Development
2. GUI
3. Business Application
4. Games and 3D Graphics
5. Accessing the Database

Extension of python is filename.py

Python code execution:

0101001--byte

Source 🡪 m.pyc(bytecode)🡪 pvm(runtime)

Installation.

Datatype: Number and string

Data🡪alphabetas

Number

Name=”online”

Num=45

Name=”a”

Numbers: integers, floating point and complex numbers

Strings: single character or group of characters.

Pre define datatypes:

Integers : 2,3,456,5678,90765….

Floating point:4.2345,67.3412….

Complex numbers: a+bj 3+4j

A=5,b=45.5,c=344567654…

Stud=”R” or “Ruth”

Name=” this is online classes of python”

Boolean:True or False

User define datatypes

List

Set

Tuple

Dictionary

Range

Variables and identifiers:

Identifiers :Any name that is used to define a class, function,variable module,or object is an identifier.

A=4

Name=”python course”

variables

Object

Classes

Modules

Functions constants

Variables:

🡪Variables or identifiers should not match with the keywords.

🡪Variable name should start with character or underscore (\_)

🡪It can be alphanumeric (characters + Digits) eg:name2

🡪variable name is case sensitive name=”Hello” Name

Comments:

# single line comment.

/\* multiple line \*/

‘’’ ‘’’

Assignment can done in multiple ways:

Sum=0

Flag=0

A=0

B=0

X=0

Sum=flag=a=b=0

Sum,flag,name,x=0,1,”World”,35.4

Reserved words:

True, False, None

and , or ,not ,is ,if else,elif,while,for,break ,continue,return, in,yield,try,except,finally,raise,assert,import,from,as,class,def,pass,global.nonlocal,lambda,del ,with

Operators:

Arithmetic operators

+,-,\*,/,//,%,\*\*

Relational operators: <,>,<=,>=,!=,==

Unary operators: Unary plus(+),unary minus(-)

Bitwise operators : |,&,^~,<<,>> eg:^ 5 => 00000101 t t🡺f and f f 🡺f ;

0101

0011

0001=1🡺&

Bitwise &

12 → 0000 1100

13→ 0000 1101

0000 1100 = 12

0111=7🡺|

Bitwise |

12 → 0000 1100

13 → 0000 1101

0000 1101 = 13

0110=6🡺^

Bitwise ^ (XOR)

12 → 0000 1100

13 → 0000 1101

0000 0001 = 1

Tilde ~

~12

12 → 0000 1100

1’complement è 1111 0011

Add 1 ⇒ 1

1111 0100

0101🡺8\*0+4\*1+2\*0+1\*1=5🡺1010

0000 1100 12<<2 12>>2

Logical operators: &&,||,! And or ! 3 => 00000011 t f 🡺t f t 🡺t

Assignment operators: =,+=,-=,\*=,/=,%= 110 🡺6

Membership operators: in and not in (eg: in variable in sequence)

Identity operators : is and is not

<< 🡪binary left shift

>> 🡪binary right shift

& 🡪 and

Right shift

4 == (0000 0100) è (0000 01)= 1

Left shift

4== (0000 0100)è (00 0100 00 )=16

111

24=() 2 24 5🡺 00000101

2 12—0 3🡺 00000011

| 2 6--0 100=4

< 2 3—0

1---1 🡺(11000)2

<=

>=

==

!=

LIST:

Lists are used to store multiple items in a single variable.

It is a collection of heterogeneous data

[6,2,4,8,5,2]

🡪mutable

🡪list items are ordered ( order is preserved)

🡪changeable : we change by add and removing items in lists

🡪duplicate are allowed

🡪 indexing is possible

🡪separated by ,

List1=[]

List2=[3,45,67]

print(type(seq))

<class 'list'>

seq.append(4)

seq

[4]

seq.append(2)

seq.append(6,5)

seq.extend([4,5,6,7])

seq

[4, 2, 4, 5, 6, 7]

list1=[5,6,4,3,7,8]

list1

[5, 6, 4, 3, 7, 8]

seq[0]

4

seq[1]

2

seq[5]

7

seq[2]=9

seq

[4, 2, 9, 5, 6, 7]

del list1[3]

list1

[5, 6, 4, 7, 8]

seq[-3]

5

seq[0:5]

[4, 2, 9, 5, 6]

seq[0:6]

[4, 2, 9, 5, 6, 7]

seq[3:]

[5, 6, 7]

seq[3:5]

[5, 6]

seq[3:4]

[5]

seq[:4]

[4, 2, 9, 5]

Built in functions in list:

* Which will return some value but doesn’t change list .

1)count() 🡺 list1.count() 🡺 no. of occurrances.

2)Index(value) 🡺 index position

To create a list of elements with squares :

Vv=[expression for variable in sequence]

Vv=[I \*I for I in range(1,6)]

Tuple: are used to store multiple items in a single variable. (collection of heterogeneous data which are separated by commas .

Tuple is a collection of data

Tuple is a collection which is ordered is a collection which ordered and unchangable.

Tuples are written with round brackets()

* It is unmutable
* It allows duplicates
* Order is preserved
* Indexing is possible
* Slicing is possible
* Round brackets

Tuple items are indexed

🡺create tuple

🡺update tuple

🡺remove tuple

🡺slicing (accessing elements)

Variable\_name=(elements)

A=(4,3,6,1)

SET set is a collection of heterogeneous data which are separated by commas enclosed in flower braces.

{5,”Priya”,78,90,89.6}

It is mutable

Duplicates are not allowed

Index is not possible

Slicing is not possible

operations on set:

issubset()

s.issubset(t)

a={1,2,6,8} b={2,3,4,9}

symmetricdifference= {1,6,8,3,4,9}

DICTIONARY:Group of (key ,value) pairs separated with a commas and enclosed in curly brackets.

Dictionaries are used to store data values in key:value pairs.

A dictionary is a collection data which is ordered , changeable and does not allow duplicates.

Dic={“Name”:”Priya”,”course”:”python”,”duration”:3,”year”:2021}

Dic={}

Changeable means we can change,add or remove items after the dictionary has been created.

Dictionaries cannot have two items with the same key

Accessing : with the help of the keys . we can acces corresponding values

Dic[“course”]

dictionary comprehension:

dic={expression iteration condition}

dic={x:x\*\*3 for x in range(5) if x%2==0}

d1.pop("time")

'2hrs'

d1

{'name': 'social', 'course': 'fullstack', 'branch': 'cse', 'place': 'hydf'}

d1.popitem()

('place', 'hydf')

b=d1.copy()

b

{'name': 'social', 'course': 'fullstack', 'branch': 'cse'}

⇒b.get('course')

fromkeys(tuple,value)

tuple1=(‘a’,’b’,’c’)

d.fromkeys(tuple1,12)

→ setdefault(key, value)

d.setdefault(‘sno’,21)

A program’s control flow is the order in which the program’s code executes.

The control flow of a python program is regulated by conditional statements , loops and function calls.

Python has 3 types of control structures.

🡪Sequential statements

🡪Decision or selection

🡪Iterative or repetition

1. Sequential statements: are a set of statements whose execution process happens in a sequence.

A=10

B=5

C=a+b

1. The selection statement allows a program to test several conditions and execute instructions based on which condtion is true.

🡺 Simple if

* If … else
* Nested .. if
* If …elif … else

If true block

Syntax:

If condition or expression: (header)

Statements 1

Statements 2

.

.

Statements n

Statements

**If .. else:**

If (condition):

Statements

Else:

statements

**nested if**

if( condition):

if(condition):

statements

else:

statements

else:

statements

if elif else

if(condition):

statements

elif condition:

statements;

else:

statements

🡺Iterative or Repetition : used for looping ie., repeating a piece of code multiple times

A repetition statements is used to repeat a group of (block) programming instructions

🡪For loop

🡪While loop

For loop: A for loop is used to iterate over a sequence that is either a list ,tuple, dictionary or a set.

For variable in sequence: (list,tuple,dictionary , range all are these accepted here)

Range (start,stop, stepsize)

→ loop variable initialization

→condition for termination

→updation of variable

While loop:

initial

While condition:

Statements

.

.

Incre/decre

Functions:

A function is a block of code which only runs when it is called.

You can pass data, known as parameters into a functions

A function can return data as a result.

🡺Reusability

🡺Easy debugging

Function declaration or definition

Function call

Rules:

Every function should start with ‘def’ keyword

Every function should have name(not equal to any keyword)

Parameters(optional) include in between paranthesis.

Every function name with/without arguments should end with(:)

Return value 🡪 empty/value (multiple return values )

def display():

statements

display()

types of arguments:

* Required arguments
* Keyword arguments
* Default arguments
* Variable arguments

1)Required arguments: number of arguments should be same in both function call and function def.

Position or order should be followed.

Def add(a,b):

Print(a,b)

add(2,3)

2)keyword arguments:position or order is not required.

Initialization will be done based on keyword(name)

Def add(a,b):

Print(a,b)

Add(b=3,a=10)

Default arguments: number of arguments not be match with the function

Def add(name,course=”fullstack”)

Print(name)

Print(course)

Add(name=“Priya”)

Add(name=“Priya”,course=”m.tech”)

Variable length arguments: Arbitrary number of arguments.

By placing \* as prefix to the arguments of function definition

Def disp(\*course):

For I in course:

Print(i)

Disp(“fullstack”,”angular”,”react”,”javascript”)

Recursion function: python also accepts function recursion .

Function calls itself is known recursion.

Base case🡪 terminating from recursion

Recursive case🡪 keep on calling itself

Def ff(n):

If(n==0 or n==1):

Return 1

Else:

Return n\*ff(n-1)

res=Ff(3)

print(res)

**Anonymous functions or LAMBDA Function:**

🡪it is name less ,definition less(function doesn’t have any name)

🡪do not defined by ‘def’ keyword

🡪defined by using lambda keyword

* It is light weight and single use
* Can not access global variables
* Return expression but not value
* One line functions

Syntax: lambda arguments or parameters:expression

Var1=lambda x,y :x+y

Print(var1)

STRING:are stored as individual characters ie. Indexed wise. The index by default start with 0.

str=Hello

01234🡪 forwarding indexing

-5-4-3-2-1 🡪backward indexing

Str[0]🡺H str[-2]🡺l

S=”Hello”

S=’Hello’

S=’’’Hello’’’

S[2]=’t’ 🡺not possible

String are immutable .Hence you can not change the individual letters of string using assignment operator.

Str=”Welcome to python classes”

Concatenation(+):

Repetition(\*)

Str.split()🡺 create list with elements of given string separated by space

FILES: are collection of data

There are two types of file

🡺text file 🡺binary file

Open--- ,close

fileobj=Open(“filename.txt”,”access mode”)

points to the content of file

Access Modes:

Read (r) 🡺file pointer points at the beginning of the file

🡺only for read the content

Write (w)🡺 only for write the content

🡺file pointer points at the beginning of the file.

🡺if the file doesn’t exist then new file will be created with given file name

🡺data is overwritten

Eg: fp=open(“file1.txt”,”w”)

Fp.write(“welcome to python file classes”)

Fp.close()

Append (a) :to be append in the existing file

🡺file pointer points at the end of the content in the file.

🡺file doesn’t exist, new fille will be created and appended.

a🡪 appending; a+🡪reading and appending

w🡪 writing; w+🡪 writing and reading

r🡪reading r+🡪reading and writing

binary files:

rb🡺reading;rb+🡺reading and writing

wb🡺writing; wb+🡺reading and writing

ab🡺appending; ab+🡺appending and reading

Binary files: filename.dat

Bytestream

Pickle🡪 module🡪dump(), load()

Structure==🡺Bytestream(file)

Fp=open(“binary1.dat”,”wb”)

Ls=[“Priya”,”Aparanjini”]

Pickle.dump(ls,fp)

Fp.close()

Fp=open(“binary1.dat”,”rb”)

V1=Pickle.load(fp)

Print(v1)

f.close()

import pickle

fp=open("bina1.dat","wb")

ls1=["priya","Aparanjini"]

pickle.dump(ls1,fp)

fp.close()

fp=open("bina1.dat","rb")

pp=pickle.load(fp)

print(pp)

fp.close()

DATABASE CONNECTIVITY: (MYSQL)

1)

* Import mysql.connector
* Establish connection (connect())
* Create cursor object
* With cursor\_object execute query

2)mysql.connector.connect(host=”localhost” ,user=”root”, passwd=”Priya”)

import mysql.connector

#co=mysql.connector.connect(host="localhost",user="root",passwd="priya")

#print(co)

#curob=co.cursor()

'''

#displaying databases

curob.execute("show databases")

for db in curob:

print(db)

'''

'''

# creating database

curob.execute("create database books")

'''

'''

co=mysql.connector.connect(host="localhost",user="root",passwd="priya",database="books")

curob=co.cursor()

curob.execute("create table kidsbooks(bid int(5) not null primary key,bname varchar(20),dprice int(8))")

'''

#insertion

co=mysql.connector.connect(host="localhost",user="root",passwd="priya",database="books")

curob=co.cursor()

query="insert into kidsbooks(bid,bname,dprice) values(%s,%s,%s)"

'''

value=(101,"kidsabc",2000)

curob.executeone(query,value)

co.commit()

'''

'''

value=[(102,"xyz",2000),(103,"Hellochild",5000)]

curob.executemany(query,value)

co.commit()

'''

'''

curob.execute("select \* from kidsbooks where bid=102")

#res1=curob.fetchall()

res1=curob.fetchone()

for i in res1:

print(i)

'''

'''

#updation

que="update kidsbooks set bname='children' where bid=102"

curob.execute(que)

co.commit()

print(" updated successfully....")

'''

#delete

q="delete from kidsbooks where bid=101"

curob.execute(q)

co.commit()

print(" deleted record...")

curob.close()

Exception:

Print(“welcome”)

A=[2,3,4,5]

A[6]🡪

5/2=2.5

5/0

Try and except statements

Finally

Try and except statement are used to catch and handle exceptions in python. Statements that can raise exceptions are kept inside the try clause and the statements that handle the exception are written inside except clause

Try:

Statements 1

Statements 2

Except:

Print(“error”)

Finally keyword in python:

The final block always executes before normal termination of try block or after try block terminates due to some exception.

Generator: (memory management)

Def sum():

List1=[4,5,6]

Return list1

Return 4

Return 5

Return 6

A=Sum()

Print(a)

Def fun1():

Yield 4

Yield 5

Yield 6

A=fun1()

Print(a)

Print(type(a))

Print(a.next())

Array:

Collections of data items

Import array : this will import the entire array module

From array import \* : this will import all class, objects,variable etc.,

Syntax:

Import array

Array\_name=array.array(‘type-code’,’[elements])

from array import \*

array\_name=array(‘type\_code’,[elements])

Typecode pythontype

‘b’ int 1

‘u’ Unicode character

‘i’ int 2

‘f’ float 4

‘d’ double 8

‘U’ Unicode string

‘m’ timedelta

‘M’ datetime

‘bool’ boolean

‘v’ void

Numpy Libray: relate to mathematical

Numerical python

Provide lot of functions to work in a domain of linear algebra ,fourier transforms and matrices.

Provide functions of related to arrays

Create an array called ndarray(ndimension array) is faster than list

Arrays with numerical ranges: returns array with values in given range and required datatype

Arange()

Linspace()

Logspace()

Np.Arrange( startindex,stopindex,stepsize,dtype)

Np.linspace(startindex,stopindex,num,endpoint,retstep,dtype)

Num: number of required values in array (optional)

Endpoint: True stop index will be included

False stop index will not be included

Retstep: different between values.(b/w elements)

Logspace:

Np.logspace(startindex,stopindex,num,endpoint,base,dtype)

Startindex and stopindex : lower n upper boundary

Num=number of required values in array

Base: default is 10

Endpoint: included endindex(true) not included endindex(false)

Dtype=int/float

Existing data:

Np.asarray(input,dtype,order)

Input: list /tuple/any combination

Dtype: int/float/s1(string(byte))

Order: row 🡪”C” default

Column 🡪 “F”

Frombuffer(buffer,dtype,count,offset)

Buffer:existing data

Dtype: type of data

Count: length default =-1

Offset:position default=0

→Traditional way array()

→Homongeneous data : ones() zeros()

→Diagonal : eye() ,identity(),diag()

→Numerial :Arange(),linspace(),logspace()(returns array with values in given range and required datatype)

→Random numbers: random(), rand(), random.randint(),random.randn()

Zeros: this function create an array(either one dimensional or multi dimensional ) and fill all the values with zeros.

Arr=np.zeros(5)

Ones: This function creates an array(either one dimensional or multi dimensional ) and fill all the values with ones

Arr=np.ones(5)

Eye:This function creates an array with all the diagonal elements as 1 and rest as 0 .

Arr=np.eye(3)

Identity: same eye only different is

Identity array is a square array with ones on the main diagonal.

Identity(n,dtype=None) (dtype is optional)

Np.identity(4,dtype=int)

Diagonal : This function creates a two dimensional array with all the diagonal elements as the given value and rest as 0.

A=np.diag([1,2,3,4])

A

Numerial :Arange(),linspace(),logspace()(returns array with values in given range and required datatype)

syntax: arange()

np.arange(startindex,stopindex,stepsize,dtype)

np.linspace(startindex,stopindex,num,endpoint,retstep,dtype)

num= number of required values in array optional

default num value is =50;

endpoint=True or False ( stop index will be included, not will be included)

retstep= True or False ( different b/w elements)

np.logspace(startindex,stopindex,num,endpoint,base,dtype)

base default =10

Random numbers:

🡺Randint:This function is used to generate a random number between a given range.

Syntax: rand(min,max,total\_values)

Np.random.randint(1,10,3) (max is not included)

🡺rand: This function is used to generated a random values between 0 to 1

Syntax: rand(number of values)

Np.random.rand(4)

🡺randn: This function is used to generate a random values close to 0(zero) .This may return positive or negative numbers

Syntax: randn(number of values)

Np.random.randn(6)

Shape: Returns tuple specifying indices and the number of elements.

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

Np.shape(a)

import numpy as np

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

np.shape(a)

np.reshape(a,(2,3))

a1=np.array([[1,2,3],[8,6,7]])

np.shape(a1)

np.reshape(a1,(3,2))

a2=np.array([[21,31],[41,51],[67,89]])

np.shape(a2)

np.reshape(a2,(2,3))

slicing array: [ : : ]

[start index:stopindex:stepsize]

A=np.array([1,2,3,4,5,6,7,8])

A[0:8:1]

A[1:3]

0 (0 (0,1 2),1(0,1,2))

1(0(0,1,2),1(0,1,2))

| 0  0   | 0   | 0  1 | 1  2 | 2  3 | | --- | --- | --- | | 1   | 0  4 | 1  5 | 2  6 | | --- | --- | --- | | | --- | --- | --- | --- | --- | --- | --- | --- | | 1  1   | 0   | 0  7 | 1  8 | 2  9 | | --- | --- | --- | | 1   | 0  2 | 1  6 | 2  8 | | --- | --- | --- | | | --- | --- | --- | --- | --- | --- | --- | --- | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

A[0,0,2:3]🡺3

A[0,1,1:2]🡺5

A[1,1,0:2]🡺2,6

Copy() : external copy created

🡪any one element is modified then it will not reflect to another

A a1

Two array

view(): create a reference to existing array

* Any one element is modified then it will be reflected on original array
* Only one array.

Joining Array:

(concatenation())

Concatenate((a,b),axis) 0🡪columns ; 1🡪 rows

Stack((a,b))

Splitting array

Split(array,no.of split)

Module:

Def hello(name):

Print(“Hello!”,name)

LOCAL AND GLOBAL VARIABLE:

Local variable are those which are initialized inside a function and belongs only to that particular function.

It cannot be accessed outside the function

Global variable are those which are not defined inside any functions and have a global scope.

Oops concepts in python(object oriented programming): it is implemented in classes and objects

🡪class

🡪object

🡪abstraction

🡪encapsulation

🡪inheritance

🡪polymorphism

1)Class: collection of variables(data) and functions

🡪Blue print which is followed by objects

🡪logical structure with behavior

Class

Attributes(data) }rno sname salary

Functions } display() read() write()

Class

Data functions

Access

Object(reference)

Syntax:

Class class\_name:

Data

.

.

Functions

.

.

.

2) Object : Instance of a class ,any number of objects

🡪object (this is physical entity)

Attributes

Functions

Syntax:

Object\_name=class\_name()

1. Abstraction:

🡪Hiding the implementation details

🡪showing essential details

1. Encapsulation:Binding data and functions into a single entity.

Public: Accessed by any functions in any class

Private: Accessed by only class

Protected: Accessed by only class or class inherited

5)Inheritance: derived class inherits the properities of base class.

Course python

Attributes attributes

Functions functions

Single inheritance:

One base class and one derived class.

Multiple inheritance:

Two base classes and one derived class

Class A class B

Class C(A,B):

Multilevel inheritance :

Baseclass A grandparent

Derivedclass B parent

Derivedclass C child

Hierarchical inheritance:

Parent class

Child \_1 child\_2 child\_3

Polymorphism: implementing same method in different content.

Methods with same name implementing in different way.

‘+’ 🡺 concatenation(string)

‘+’ 🡺 Arithmetic (addition)

Overloading: we can define a method in such a way that there are multiple ways to call it.

Given a single method or function . we can specify the number of parameters ourself.

Override : means having two methods with the same name but doing different tasks

It means that one of the methods override the other method

If there is any method in the super class and a method with the same name in a subclass

Special variable in python: \_\_name\_\_

Abstract: A class that consist of one or more abstract method is know as abstract class. Abstract method do not contain their implementation(Abstract Base Classes)

Def sum():

/C= A+b

Print(c)/

Abstract class can be inherited by the subclass and abstract method get its definition in the subclass.

Python provides the “abc” module to use abstraction in the python program.

From abc import ABC

@abstractmethod

We use the @abstractmethod decorator to define an abstract method.

🡪abstract class :which contain only abstract methods

🡪abstract method: method with declaration but not the definition

🡪concrete class: A class without abstract methods

🡪object can’t be instantiated for abstract class

🡪object can only be instantiated for concrete class.

Regular expression: It is a sequence of characters used for pattern matching.

Pattern:It can be a character or group of characters.

RegEx—module re

Package to use for Regular expression

Import re

Basic functions:

🡪findall() 🡺return list of all matches

🡪search() 🡺 return match object if there are any matches.

Matchobject have different methods:

🡪start()🡪 give position of occurrence

🡪span()🡪 tuple of start and end position of match

🡪string() 🡪return the actual string used for pattern matching

🡪split()🡺split the string from the given pattern

🡪sub() 🡺 substitute new string to old string one or multiple.

🡪re.findall(“pattern”,source\_string)

🡪re.search(“pattern”,str)

🡪re.split(“pattern”,sourcestring,maxsplit)

🡪re.sub(“pattern”,”newpattern”,sourcestring,no.ofoccurrances)

Metacharacters -special /sequences – sets

[]🡪returns a match if string contains pattern/characters specified in []

^🡪returns a match if string starts with given pattern

$ 🡪 ends with

. 🡪any characters except new line

\*🡪zero or more occurrances

+🡪one or more occurrence

{}🡪specified number of occurrances

/🡪 special sequences.

\d 🡪 if given string having digits (0-9)

\D🡪 if given string doesnot have digits

\w🡪 if given string having word characters

\W 🡪 if given string doesnot have word characters

\s🡪if given string having spaces

\s🡪 if given string ends with specified characters

Date and time module:

Date : year,month n day

Time: hours,minutes,seconds and microseconds

Current: year month day hours mins secs micros

Import datetime

Datetime🡪 datetime – class

Datetime.datetime.now()

Datetime(7 parameters) 🡪 7fields

To format date and time

Strftime(parameters )

Year : 2 digits %y ; 4 digits %Y

Month: short %w ; long %W (full)

Day : %a; A

Hours: %H 24hrs; %I (0-12)

Minutes: %m 0-59

Seconds %s 0-59

Microseconds :%f 000000 to 999999

timeDelta:

timedelta():🡪 manipulate the date

collections in python:

what are collections:

There are four collection data types in python which are used to store collections of data.

Lists 🡪 mutable []

Tuple 🡪 immutable()

Sets 🡪 ordered not indexed n not duplicate{}

Dictionary 🡪 { keys , values}

What is a collection module in python?

Specialized collection data structures

Chainmap()

Namedtype()

Deque()

Counter()

ordredDict()

defaultdict()

chainmap: is a dictionary like class for creating a single view of multiple mappings from collections .

import ChainMap

a={‘1’:’HTML’,2:’CSS’ }

b={3:’Javascript’,4:’python’}

res=ChainMap(a,b)

namedtuple():returns a tuple with a named value for each element in the tuple

d=(name=’priya’,course=’python’,course1=’django’)

a=namedtuple(course,name,

counter: is a dictionary subclass, for counting hasable objects

a=[1,2,3,1,2,3,4,5,4,6,7,6,7,7,8,7,3,2,1]

OrderedDict: is dictionary subclass which rememners the order in which the entries were done.

Od=collections.orderedDict()

Od[‘a]=3

Od[‘r’]=4

Defaultdict: is a dictionary subclass which calls a factory function to supply missing values.

D=defaultdict(int)

D[‘p’]=4

Deque: deque pronounced as ‘deck’ is an optimized list to perform insertion and deletion easily

A=[‘p’,’r’,’I’,’y’,’a’]

Pandas: (library)(series,DataFrame and panel) data structures

🡪High performance data analysis tool

🡪large data set

🡪represents in tabular form (rows and columns)

🡪working on missing data

🡪merge and join two different datasets easily

🡪reshape datasets

Three data structures

🡺Series : one dimensional (eg:list)

🡺DataFrames : two dimensional (eg:list,dict,series,etc)

🡺Panel: MultiDimensional

Import pandas as pd

Pd.Series(data,index)

Pd.DataFrame(data)

Panel: A panel data contains observations on multiple entities(individuals) where each observed two or more point in time.

Pd.Panel(data=None,items=none,major\_axis=None,minor\_axis=,copy=False,dtype=None)

A - (65)10 - ()2

2 65

2 32—1

2 16—0

2 8—0

2 4 –0

2 2 –0

1—0

(01000001)2

Octal

0 -7

000 – 0

001 –1

010 –2

(011 ) =0\*(22)+1\*(21)+1\*(20)–3 = 0+ 2 +1=3; 20  =1 21= ,2 22=4

100 –4

101 –5

110 –6

111 –7

Hexadecimal

0000 --0

0001—1

9

10 –A

11-B

12 -- C

13 –D

14 –E

15 –F

And

| A | B | o/p |
| --- | --- | --- |
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| F | T | F |
| F | F | F |

| A | B | o/p |
| --- | --- | --- |
| 1 | 1 | 1 |
| 1 | 0 | 1 |
| F | T | T |
| F | F | F |

(12)10=(0000 1100)2

11

1’s 🡪 11110011

1

2’s🡪 1111 0100

0000 1011

2==10

10<<4

11 101000

0000 10.10

0000 11 . 11

Swap

A=3

B=5

Temp=a

A=b

B=temp

Swap(a,b)

A=a+b

B=a-b

A=a-b

A=5

B=3

A,b=b,a

Reserved words or keywords :

True , False, None

And ,or ,not ,is,if ,else, elif,while,for, break, continue, return,in, yield,try, except,finally,raise,assert, import, from, as, class, def, pass,global,nonlocal, lambda,del , with

Not allows in python

Do – while

switch

input function outfunction

input() print()

a=input(“enter number”) a=10

print(a)

print(“Hello”)

z=(a>b):”greater”?”lesser” a+=1

operators:

Arithmetic operators: +,-,\*,/,//,% ,\*\*

Relational operators: <,> ,>=,<=,!=,==

Logical operators: and ,or, not

Unary operators: unary plus,unary minus

Assignment operators: =,+=,-=,\*=,/=,….

Membership operators : in and not in

Identity operators: is and is not

Bitwise operators

| bitwise Or 🡪 both are false 🡪false

& bitwise and -> both are true 🡪 true

^ bitwise XOR 🡪both are true/ false 🡪 true

Otherwise false

~ complement 🡪 true🡪 false;false🡪 true

<< 🡪 leftshift

>> 🡪 right shift

A=5 🡪 0000 0101 🡪1111 1010

B=3 🡪 0000 0011

or 00000111 🡪7

and 00000001 🡪1

XOR🡪0000 0110

Membership operators:

Decision control structures:

If

If else

Nested if

If elif else

If(a>b)

{

Document.write(“a”);

}

If (a>b):

Print(“a > b”);

Print(“if true block”);

Print(“out of the if”)

Statement1

If(condition /expression):

Statement1

Statement2

.

.

Else:

Statement1

.

.

Statement2

Nested if:

If (condition):

If(condition):

Statement1

Statement2

Else:

Statement1

Else:

If():

Statement1

Else:

Statement1

or

Elif (condition):

Statement1

Else:

Sequential eg., 5 to 25

Decision if ,if else ,if .. elif .. else.

Iterative

Loop variable initialization

Condition for termination

Updation of loop variable

🡪While

🡪for

Syntax : while

While condition:

Statement1;

Statement2;

.

.

Statement1

Syntax:

For variable in range or sequence

Range (initial,stop,step)

Range(0 ,10,2)

Sequence list,tuple,dictionary

List:

[]

Seq=[2,6,7,8,9,4,5,3,4,2]

Seq.count(2)🡺2

Seq.index(7)🡺1

Seq.index(4,6)🡺

Tuple = ( )

Create tuple

Update tuple

Remove tuple

Slicing

Tuplevariable =(elements)

Lambda function or Anonymous function

* Do not defined by def keyword
* Return expression but not value
* One line function
* Any number of arguments
* Can not access global varaibles
* Defined by using lambda keyword
* Function doesn’t have any name
* Syntax lambda arguments : expression;

Sum=lambda x,y :x+y

🡺it is light weight

V3=lambda x: lambda y

V3(2) =()

Functions

Types of Arguments

🡪Required Arguments

🡪Keyword Arguments

🡪Default Arguments

🡪Variable length Arguments

1. 🡪Number of arguments should be same in both function call and function def.

🡪Position order should be followed

def add(a,b):

return a+b

print(add(4,6))

2)Keywords Arguments : order /position not required

def add(x,y):

return (x+y)

add(y=5,x=4)

3)default Arguments:

Default Arguments :Number of argument need not be match with the function cal;ing and function def

def disp(name,course="M.tech"):

print(name)

print(course)

disp("Aparanjin")

disp("Ganga","M.pil")

def display(\*title):

for i in title:

print(i)

display("website","django","javascript","MOngoDB")

def d1(name,\*course):

for i in course:

print(i)

print(name)

d1("Priya","HTML","CSS","JS")

def person(name,\*\*data):

print(name)

print(data)

person("priya",rollno=21,course="fullstack",fees=35000)

**recursion**

function calling itself.

Basecase 🡪 for terminating from recursion function

Recursive case🡪 keep on calling itself

def fact(n):

if n==0 or n==1:

return 1

else:

return n\*fact(n-1)

print(fact(5))

local scope:A variable which is defined inside a function is local to that function . It is accessible from the point at which it is defined until the end of the function and exists for as logn as the function is executing.

Global scope: A variable which is defined in the main body of a file is called a global variable. It will be visible throughout the file, and also inside any file which imports that file.

s=10

def s1():

p=15

print(p)

print(s)

def s2():

print(p)

print(s)

s1()

s2()

Files:

Collections of data

1)text files 2) binary files

fobj=Open(“filename”,”accessmode”)

fobj🡪 fileobject ,which point to the content of file.

Access mode:

R: readmode

Only reading the content,file pointer points at the beginning of the file.

F1=open(“ff.txt”,”r”)

W:writing mode

Only writing the content ,

A:append mode

R+,w+,a+

Rb,wb,ab,rb+,wb+,ab+

DATABASE CONNECTIVITY:

In order to create a db connectivity between python and mysql we first need to install mysql connector. Mysql connector is used to connect python with mysql so that we can store / retrieve the data from python as front end and mysql as backend.

1)import mysql.connector

2)establish connection using connect()

3) create cursor object

4) execute the query

Oops:

🡪Class

🡪Object

🡪Abstraction

Encapsulation

Inheritance

Polymorphism

🡪Class: attributes and functions

Logical structure with behavior,

Blueprint which is followed by objects



Class🡪data ,functions-🡪access🡪reference(object)



🡪Object: it also have attribute & function

🡪 Abstraction:Hiding the implementations part & show essential details

🡪Encapsulation: Wrapping of data,

Binding dats & functions into an single entity

Public 🡪 Accessed by function in any class

Private🡪Accessed by only class

Protected🡪Accessed only class or class inherited

🡪inheritance: derived class inherits the properties of base

Class pp(base or parent or super)

Class cc(derived or child or sub)

🡪polymorphism:implementing same method in different content

Methods with same name implementing in different way

Multiple inheritance:







Multilevel inheritance









Hierarchical inheritance







#single inheritance

'''

class parent1:

def parentfunc(self):

print("parent function...")

class child1(parent1):

def childfunc(self):

print("child function .....")

p=parent1()

p.parentfunc()

c=child1()

c.childfunc()

c.parentfunc()

#p.childfunc()

'''

#

'''

class A:

def \_\_init\_\_(self,a):

print("A constructor",a)

self.a=a

def disp(self):

print("display")

class B(A):

def \_\_init\_\_(self,a,b):

super().\_\_init\_\_(a)

print("B constructor...",b)

self.b=b

class C(B):

def \_\_init\_\_(self,a,b,c):

super().\_\_init\_\_(a,b)

print("c constructor..",c)

self.c=c

a1=A(3)

b1=B(4,6)

b1.disp()

c1=C(2,3,4)

'''

class Person:

def \_\_init\_\_(self,n):

print("person constructor",n)

self.name=n

class Employee(Person):

def \_\_init\_\_(self,i,n):

print("employee constructor",i)

super().\_\_init\_\_(n)

emp=Employee(12,"Aparanjini")

#single inheritance

'''

class parent1:

def parentfunc(self):

print("parent function...")

class child1(parent1):

def childfunc(self):

print("child function .....")

p=parent1()

p.parentfunc()

c=child1()

c.childfunc()

c.parentfunc()

#p.childfunc()

'''

#

'''

class A:

def \_\_init\_\_(self,a):

print("A constructor",a)

self.a=a

def disp(self):

print("display")

class B(A):

def \_\_init\_\_(self,a,b):

super().\_\_init\_\_(a)

print("B constructor...",b)

self.b=b

class C(B):

def \_\_init\_\_(self,a,b,c):

super().\_\_init\_\_(a,b)

print("c constructor..",c)

self.c=c

a1=A(3)

b1=B(4,6)

b1.disp()

c1=C(2,3,4)

'''

'''

class Person:

def \_\_init\_\_(self,n):

print("person constructor",n)

self.name=n

class Employee(Person):

def \_\_init\_\_(self,i,n):

print("employee constructor",i)

super().\_\_init\_\_(n)

emp=Employee(12,"Aparanjini")

'''

from abc import ABC,abstractmethod

class pers(ABC):

@abstractmethod

def area(self):

None

def res(self):

print("result")

class emp(pers):

def \_\_init\_\_(self,n):

self.n=n

def area(self):

print(self.n\*self.n)

p1=emp(5)

p1.area()

🡪Abstract class and abstract methods

Abstract class which contains only abstract methods

Abstract method :- method with declaration but not the definition

Concrete class:= A class without abstract methods

Object can’t be instantiated for abstract class

Object can only instantiated for concrete class.

Concrete class inherited from abstract class

Polymorphism:

Overloading Overriding

Operator overloading (+) 🡪 concatenation, Addition

Method overriding : method name should be same and arguments must be different

Sum(3,4)

Sum(4,5,6)

Regular Expression : It is a sequence of characters used for pattern matching

RegEx🡪package to use regular expression

Import re

re.findall()🡺return list of all matches

re.findall(“pattern”,source\_string)

re.search()🡺 return matchobject if there are any matches

re.split() 🡺 split the string from the given pattern

re.sub()🡺 substitute new string to old string one or multiple

search object

start()🡺 give the position of occuuances

span()🡺 tuple of start and end position of match

string()🡺return the actual string used for pattern matching

metacharacters – special /sequences - sets

[]🡺

^🡺

$ 🡺

. 🡺

\*🡺

+🡺

{}🡺

/🡺

Special sequences

\d🡺(0-9)

\D🡺

\w🡺

\W🡺

\s🡺

\S🡺

\z🡺

Sets

[abc]🡺

[a-z]🡺

[^abc]🡺

[5673]🡺

[1-9]🡺

[0-9][0-9]🡺

Date and time module

Date and time module:

Import datetime 🡺datetime class

Now()

Datetime(7 parameters)

datetime.now(x,x,x)

X=datetime.datetime(2021,10,12,12,35,45,000513)

Format:

Year %y🡪 2digits; %Y 🡪4 digits

Month %w 🡪 %W

Day %a 🡪 %A

Hours %H 🡪24hours; %I🡪0-12 ;%p🡪Am/Pm

Minutes %m 0-59

Seconds %s

Microseconds 000000 to 999999

🡺String format:

🡪Strftime(parameter)

Timedelta

Timedelta()🡺 manipulation on dates

Timedelta(days=value,hours=values…..)

Library:

Pandas: (library)

High performance Data Analysis Tool

Working with large data set

Load files with different formats and flexible

Represents in tabular from (way) (rows and columns)

Working on data

Merge and join two different datasets eaily

Reshape datasets

Data structures

Series : one dimensional eg:list

Data frames : Two dimensional eg: list/dict/series

Import pandas as pd

Pd .Series(data,index)

list1=[1,2,3,4,5]

pd.Series(list1)

pd.Series(list1,index=['a','b','c','d','e'])

pd.Series(['apple','mango','grapes'],index=['x','y','z'])

Pd.DataFrames(data)

d={'name':['sangeetha','sunitha','priya'],'course':['python','django','angular']}

c1=pd.DataFrame(d,columns=['course','name'])

c1

c2=pd.DataFrame(c1,columns=['a','b'])

c2

d1={"aaa":{"sem1":[67,78,69,87],

"sem2":[98,78,86,86]},

"bbb":{"sem3":[56,67,87,98],

"sem4":[69,59,78,96]}

}

d1

d1\_df=pd.DataFrame(d1)

d1\_df

pd.DataFrame(pd.Series(["Apple","Grapes","Orange","Banana"]))

import numpy as np

sd=pd.DataFrame(pd.Series(np.arange(10,17)))

sd

dic1={"ones":pd.Series([1,11,111,1111]),

"even":np.arange(2,10,2),

"fruits":["Mango","kiwi","Apple","Orange"]

}

dic1

D={‘name’:[‘xxx’,’yyy’,’zzz’],’course’:[‘python’,’django’,’angular’]}

Pd.Dataframe(d)

Arrow🡪

Bokeh🡪

Beautifulsoap🡪

Cirq🡺

Django🡺

Eli5🡺

Flashtext🡺

textBlob

pillow🡺pil(python image library)

wxpython🡺

web2py🡺

numpy🡺

pandas🡺

tensor flour

scikit

idle🡺default ide

jupyternotebook

spyder

pycharm

pydev

wing

atom

DJANGO :



User 🡪









Python object



Sql statement 



Urls.py

Forward request to appropriate view

Read/writedata HttpResponse Template(filename.html) 



User🡪django🡪views🡪









View.py

response

browser request url.py

Views

db Model templates

sql orm python