**Day1:**  **Course Content:**

application ==> kind of software

phonepe

banking transactions

e bills

texting

python

ecommerce apps

health care

banking apps

salesforce apps

Artificial intelligence etc.

python full stack

=================

APP

3-LAYERS

FRONT END LAYER ==> LAYER-1/TIER1

UI (USER INTERFACE) LAYER

COMMON PEOPLE

TECHNOLOGIES: HTML, CSS, JAVASCRIPT, JQUERY, BOOTSTRAP, ANY JS LIBRARY

{REACT, ANGULAR, NODE}

APPLICATION PROCESSING LAYER\

BACK END LAYER\

full stack:

==========

1) front end

2) Python

core python

python fundamentals ==> datatypes, operators, control statements

data structures ==> collections ==> list, tuple, set, dictionary etc.

sorting, searching, insertion etc.

functions

named functions ==> def

nameless functions ==> lambda, filter(), map() & reduce()

OOPs ==> CLASS, OBJECT, METHOD, INHERITANCE, ENCAPSULATION, POLY MORPHISM,

OVERLOADING, OVBERIDING, ABSTRACTION ETC. ==> BANKING APP

FILE HANDLINGS

EXCEPTION HANDLING

MULTI-THREADING

DATABSE PROGRAMMING: SQLITE3

advanced python

PYTHON LIBRARIES : NUMPY, PANDAS ETC.

3) DATABSE PROGRAMMING

=======================

SQL ==> MySQL, ORACLE, MONGODB

4) FRAMEWORKS

DJANGO

NODEJS, ANGULAR, REACT JS

5) AWS

=================================

COURSE DURATION: 3 TO 4 MONTHS

FEE STRUCTURE: 18000

TIMING: 1.30 TO 2 HRS

DAILY PRACTICE

ASSESSMENTS

PLACEMENT ASSISTANCE

DEDICATED LAB

**Day2: Programming Fundamentals:**

CORE PYTHON CONTENT

=====================

1) PROGRAMMING FUNDAMENTALS

SOFTWARE

APPLICATION

TYPES OF APPS

SOFTWARE DEVELOPMENT STEPS

TYPES OF PROGRAMMING LANGUAGES

TRANSLATORS

PROGRAMMING FUNDAMENTALS

2) PYTHON FUNDAMENTALS

ABOUT THE PYTHON:

FEATURES

APPLICATIONS

HISTORY

PYTHON SETUP

IDE SETUP

RESERVED WORDS

IDENTIFIERS

VARIABLES

COMPILE TIME DEFINITION

RUN TIME DEFINITION ==> input()

MEMORY HIERARCHY IN PYTHON

DATATYPES (PRIMITIVE)

3) IO OPERATIONS

INPUT OPERATION ==> input()

OUTPUT OPERATION ==> print()

4) OPERATORS

OPERAND

OPERATOR

TYPES OF OPERATORS ==> UNARY & BINARY

ARITHMETIC OPERATORS

ASSIGNMENT

COMPOUND OPERATOR

RELATIONAL

LOGICAL

BITWISE

SPECIAL

IDENTITY

MEMBERSHIP

5) CONTROL STATEMENTS

CONDITIONAL STATEMENTS

if, elif and else

Note: No switch statement

LOOPING STATEMENTS/ITERATIVE STATEMENTS

for and while

Note: do-while (not support)

LOOP CONTROL STATEMENTS

break

continue

practice with number problems

PATTERNS

6) DATA STRUCTURES

COLLECTIONS

STRING - OPERATIONS

LIST - OPERATIONS

TUPLE - OPEREATIONS

SETS - OPERATIONS

DICTIONARIES

PRE-DEFINED METHODS

OTHER COLLECTIONS

FROZEN SETS

BYTES

BYTEARRAY

SEARCHING TECHNIQUES

SORTING TECHNIQUES

INSERTION

PRACTICE

7) FUNCTIONS

FUNCTIONS WITHOUT PARAMETERS

FUNCTIONS WITH PARAMETERS

FUNCTIONS WITHOUT RETURN

FUNCTIONS WITH RETURN

TYPES OF ARGUMENTS

ACTUAL

FORMAL

WAYS FOR PARAMETERS/ARGUMENTS DEFINITIONS

KEYWORD ARGUMENTS

POSITIONAL ARGUMENTS

DEFAULT ARGUMENTS

VARIABLE LENGTH ARGUMENTS

TYPES OF VARIABLES

LOCAL

GLOBAL

8) ADVANCED FUNCTIONS

NAMELESS FUNCTIONS ==> lambda

FUNCTION TO FUNCTION

FUNCTION AS AN ARGUMENT TO ANOTHER FUNCTION

filter(), map() and reduce()

FUNCTION ALIASING

NESTED FUNCTIONS

9) GENERATORS & DECORATORS

10) OOPs ==> OBJECT ORIENTED PROGRAMMING SYSTEM

SECTION-1:

==========

CLASS DEFINITION

CLASS WITH ATTRIBUTES

CLASS WITH METHODS

CLASS WITH BOTH

OBJECT CREATION FOR CLASS

MULTIPLE OBJECTS FOR ONE CLASS.

ACCESSING OF CLASS DATA AND METHODS USING OBJECTS

` METHODS IN CLASS

WITHOUT PARAMETERS

WITH PARAMETERS

WITHOUT RETURN

WITH RETURN

CLASS VARIABLES

CONSTRUCTOR

WITH PARAMETERS

WITHOUT PARAMETERS

DESTRUCTORS

SECTION-2:

=========

ENCAPSULATION

OVERLOADING

OVERRIDING

INHERITANCE

POLYMORPHISM

SECTION-3:

=========

DATA ABSTRACTION

STATIC VARIABLES

PRIVATE VARIABLES

PUBLIC VARIABLES

MINI PROJECT ==> BANKING APPLICATION

11) FILE HANDLING

CREATE A FILE

OPEN THE FILE

WRITE INTO FILE

\ READING FROM FILE ETC.

12) EXCEPTION HANDLING

13) MULTI THREADING

14) DATABASE PROGRAMMING

SQLITE3

15) REGULAR EXPRESSIONS

=======================================================

1) PROGRAMMING FUNDAMENTALS

==============================

SOFTWARE

==========

==> multi-tasking

Ex: windows ==> operating system

ms-excel ==> application

ms-word

paint etc.

==> a collection of programs

APPLICATION

============

==> app

==> running with multiple programs

==> small software

ex: whatsapp ==> texting{incoming, outgoing}, audio calling {incoming, outgoing}, ...

Mobile software ==> Android or IoS

operating system:

===============

core software/main software

which it can run small softwares/applications

==> two types:

1) standard ==> desktops, laptops ==> ex: windows, Linux, mac-os

2) non-standard ==> smart phones/tablets ==> ex: android, IoS etc.

TYPES OF APPS

==============

working domains: banking, ecommerce, health care, insurance, telecom etc.

==> classified into 3-types:

1) web applications ==> with internet

2) standard apps/desktop apps ==> run on standard os

3) mobile applications ==> run on non-standard os

**Day3: Software Setup**

SOFTWARE SETUP

================

1) CHECK WHETHER THE PYTHON SOFTWARE IS ALREADY INSTALLED OR NOT IN OUR SYSTEM.

procedure:

open the command prompt ==> type a command: python --version

2) WHEN THE PYTHON SOFTWARE IS NOT INSTALLED:

i) DOWNLOAD THE PYTHON SOFTWARE

PROCEDURE:

open the browser ==> search : python.org ==> click on : downloads ==> from downloads page, we can required to download the corresponding software.

(or)

Instead of above navigatioin steps:

use this below link:

https://www.python.org/downloads/

according to the system configurations: we should do the downloading of python software.

ii) WE SHOULD INSTALL THE DOWNLOADED PYTHON SOFTWARE FILE.

i) click on the downloaded file

ii) select: add file .py to PATH

iii) click on: install mow

iv) click on: close

devices ==> we are using ==> can be operated with three kinds of OS:

1) windows

2) Linux/Unix

3) mac-Os

IDE (INTEGRATED DEVELOPMENT ENVIRONMENT)

==========================================

==> A THIRD-PARTY SOFTWARE

==> WE CAN USE:

1) TO BUILD OR DEVELOP OR WRITE LARGER AMOUNT OF CODE

2) TO COMPILE THAT LARGER CODE

TO FIND ERRORS IN A PYTHON CODE

3) TO RUN THE LARGER CODE

TO GET THE OUTPUT

==> THE POPULAR IDEs FOR PYTHON ARE:

1) VS CODE (VISUAL STUDIO CODE)

2) SUBLIME TEXT

3) PYCHARM

4) ECLIPSE

5) INTELLIJ ETC.

VS CODE INSTALLATION

======================

1) WE REQUIRED TO DOWNLOAD THE VS CODE.

browser ==> search : VS code download ==> you can download according to your system configurations

(or)

use the below link to get download the VS code:

https://code.visualstudio.com/download

2) AFTER THE DOWNLOADING: WE SHOULD INSTALL THAT DOWNLOADED FILE.

i) click on downloaded file

ii) select: Agree the conditions

iii) click on: next

iv) click on: next

v) click on: next

vi) select: create a desktop shortcut

vii) click on: next

viii) click on: install

ix) click on: finish

PYCHARM ISNTALLATION

======================

1) DOWNLOADING OF PYCHARM

chrome ==> search about: pycharm download for windows ==>

or

go through this link:

https://www.jetbrains.com/pycharm/download/?section=windows

PYCHARM PROFESSIONAL ==> WITH 30 DAYS OF FREE TRAIL

PYCHRAM COMMUNITY EDITION ==> GENERAL PYTHON PRACTICE ==> OPEN SOURCE AND FREE

BASED ON THE TYPE OF SYSTEM PROCESSOR, WE SHOULD DOWNLOAD THE SOFTWARE.

2) AFTER THE DOWNLOADING:

WE SHOULD INSTALL THE SOFTWARE:

i) run the downloaded file by simply click on that downloaded file

ii) click on: yes

iii) click on: next

iv) click on : next

v) select two options:

a) create desktop shortcut

b) add bin to path

vi) click on: next

vii) click on: install

**Day4:**  **Python Programming Fundamentals Part\_01**

IDE OPERATIONS

===============

HOW TO CREATE A FOLDER

========================

IN VS CODE

==========

CREATE A FOLDER IN WINDOWS OF ANY DRIVE

RIGHT CLICK OF MOUSE ==> NEW ==> FOLDER ==> ENTER THE TITLE FOR THE FOLDER

SET THE BACKGROUND THEME:

=============================

SETTINGS ==> THEMES ==> SELECT YOUR THEME

IN PYCHARM:

============

MAIN MENU ===> FILE ===> NEW PROJECT ===> ENTER YOUR PROJECT NAME ===> CLICK ON: CREATE

ADDING OF SUB FOLDER TO THE PROJECT FOLDER:

=-============================================

SELECT THE PROJECT FOLDER ===> RIGHT CLICK OF MOUSE ===> NEW ===> DIRECTOR ===> TYPE A NAME FOR THE SUB-FOLDER ==> CLICK: ENTER

TO CHANGE THE BACKGROUND THEME:

==================================

CUSTOMIZE ===> APPEARANCE ===> THEME ===> LIGHT

TO CHANGE THE FONT:

====================

CUSTOMIZE ===> ACCESSABILITY ==> ADJUST THE FONT

HOW TO CREATE A PYTHON FILE

=============================

A PYTHON FILE ==> EXTENSION ==> .py

SYNTAX:

filename.py

ANY PROGRAMMING FILE ==> SOURCE FILE

IN C: .c file

IN C++: .cpp file

IN JAVA: .java file

IN PYCHARM:

==========

SELECT THE FOLDER ==> RIGHT CLICK OF MOUSE ===> NEW ===> PYTHON FILE ===> ENTER THE FILE NAME WITHOUT ANY EXTENSION

IN VS CODE

==========

FILE ===> NEW FILE ===> ENTER YOUR FILE ===|> CLICK ON ENTER ===> SELECT FOLDER ===> CLICK ON OKAY

=======================================================================

FIRST PYTHON PROGRAM

======================

"HELLO WORLD" PROGRAM

FEATURES OF PYTHON:

====================

1) EASY PROGRAMMING LANGUUAGE

COMPARING WITH OTHER LANGUAGES, WE CAN DO ANY DEVELOPMENT WITH PYTHON USING MINIMAL CODE

2) INTERPRETER DEPENDENT LANGUAGE

PYTHON : FOR THE DEVELOPMENT & FOR THE SCRIPTING (TESTING)

3) GENERAL PURPOSE PROGRAMMING LANGUAGE

PYTHON:

WORKING DOMAINS:

ECOMMERCE, HEALTHCARE, BANKING, INSURANCE ETC.

WEB APPLICATIONS, DESKTOP APPS, MOBILE APPS ETC.

4) OBJECT ORIENTED PROGRAMMING LANGUAGE

5) HIGH LEVEL PROGRAMMING LANGUAGE

USING SOME USER UNDERSTANDABLE WORDS/KNOWN WORDS ==> PYTHON CODING

ex:

if a > b:

big = a

else:

big = b

classified into:

` three types:

1) structured ==> data structures{arrays, structures, pointers, unions, linked list} ==> Ex: C, C++, Python

2) object based ==> when the development can be implemented with only: class and object ==> Ex: VB

3) object oriented ==> support all the features of OOPs ==> Ex: Java, Python, C# AND c++ ETC.

concise and secure applications

OOPs ==> Object Oriented Programming System

class, object, methods, constructors, destructors

modules, packages,

Encryption, Inheritance, Polymorphism, overloading

overriding, abstraction etc.

6) STRUCTURED PROGRAMMING LANGUAGE

LIST, TUPLE, SETS, DICTIONARY, BYTES AND BYTEARRAY ETC.

7) OPEN SOURCE

PYTHON ==> WINDOWS, LINUX AND MAC-OS

8) FREE SOFTWARE

9) DYNAMICALLY TYPED PROGRAMMING LANGUAGE.

CLASSIFICATION OF HIGH-LEVEL PROGRAMMING LANGUAGES

======================================================

BASED ON THE DEFINITION OF THE DATA:

==> THREE TYPES:

1) STATICALLY TYPED PROGRAMMING LANGUAGE ==> C, C++

2) STRONGLY TYPED ==> JAVA

Syntax for the data definition:

type\_of\_the\_data name\_of\_the\_data = value;

3) DYNAMICALLY TYPED ==> PYTHON

==> for the data definition:

not require the type of the data

based on the assigned value, the type of the value can be detected by the python automatically

type()

====

==> pre-defined method

used to get the type of the data

Syntax:

type(value/name\_of\_value)

==> not a writable method

==> readable method

Note:

====

type() will provide the type of the single value/object at a time.

print()

====

==> output method or write method

used to print anything on the screen.

Syntax:

print(data)

print(d1,d2,d3,..)

a = 100

b = 123.234

c = "Ravi"

print(type(a))

print(type(b))

print(type(c))

print(a)

print(b)

print(c)

print(a,b,c)

**Day5:** **Python Programming Fundamentals Part\_02**

keywords in python

=================

==> pre-defined words

which have a specific meaning

==> are also called as "reserved words".

==> total of 35 keywords in python.

==> for the keywords there is a module (pre-defined) with the name "keyword" in python library

an identifier to identify all 35 keywords from "keyword" module "kwlist"

ex:

# to find the smallest among two numbers

a = 97

b = 79

if a < b:

small = a

else:

small = b

================================

# WRITE A PYTHON PROGRAM TO LIST AND DISPLAY ALL AVAILABLE KEYWORDS.

import keyword

keyword\_list = keyword.kwlist # listing all the available keywords

print(keyword\_list) # printing of all the available keywords as a group.

['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def', 'del',

'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or',

'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']

=====================================================================

COMMENTS IN PYTHON

====================

==> most valuable part of the program

==> the comments are some extra-lines of our program

which are not to be executed.

==> comments can be used for the "documentation". it can be useful for the reading purpose.

==> to add the description about the program ==> comments are used.

==> COMMENTS CAN BE ALLOWED TO ADD IN ANYWHERE OF THE PROGRAM.

==> two ways to write comments:

1) using # ==> the comments can be within the same line. (single line comments)

2) using triple quotes ==> when the comments in multiple lines

# WRITE A PYTHON PROGRAM TO LIST AND DISPLAY ALL AVAILABLE KEYWORDS.

import keyword

# keyword list = keyword.kwlist listing all the available keywords

#

# print(keyword\_list) printing of all the available keywords as a group.

print(keyword.kwlist)

"""

first approach:

1. we need to import the keyword module from python library using "import" keyword.

2. we required to prepare all the keywords as group using 'kwlist' is an identifier and 'keyword' is a module.

3. print that required or prepared group using print()

"""

'''

second approach:

1. we need to import the keyword module from python library using "import" keyword.

2. we should create all keywords as a group and print that group using print()

'''

============================================================

IDENTIFIERS IN PYTHON

=====================

==> WHEN WE REQUIRED TO NAME ANY ENTITY IN THE PROGRAM, WE USED "IDENTIFIERS".

HERE, ENTITIES OF PROGRAM ARE:

VARIABLES ==> FOR THE DATA STORAGE.

FUNCTIONS ==> MODULAR APPROACH

CLASS ==>

OBJECT, METHODS, MODULES, PYTHON FILE ETC.

RULES FOR AN IDENTIFIERS:'

=========================

1) THE ALLOWED CHARACTERS:

alphabets ==> upper case/lower case/both

digits ==> 0 to 9

only one special character ==> \_ (underscore)

ex: ravi\_27, python, etc ==> valid identifiers

2) THE IDENTIFIER SHOULD NEVER START WITH DIGIT.

EX: 97abc, 123ravi etc. ==> invalid

Note:

if the identifier with the digit at the beginning ==> Syntax Error

Note:

if the identifier with underscore at the start ==> valid

3) THE IDENTIIFER NEVER BE INCLUDE WITH SPACE.

Note: if an identifier with space ==> Syntax Error.

4) THE IDENTIFER NEVER ALLOWED WITH ANY OTHER SPECIAL CHARACTERS.

Note: if an identifier with a special character ==> Syntax Error.

5) IDENTIFERS ARE CASE SENSITIVE.

EX: A NAME WITH LOWER CASE AND SAME NAME WITH UPPER CASE, BOTH ARE CONSIDERED BY THE INTERPRETER AS DIFFERENT.

ex: abc = 100

ABC = 200

6) NO LIMIT IN LENGTH OF AN IDENTIIFER.

abc = 100 # valid identifier

abc\_123 = 1000

# 123abc = 10000

\_123 = 10000

\_abc = 9797

# python program = 121212

# abc@12 = 1234565

Abc = 1212

ABC = 2121

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA = 123456789

print(\_123)

print(abc)

print(abc\_123)

# print(123abc)

print(\_abc)

# print(python program)

# print(abc@12)

print(Abc)

print(ABC)

print(AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA)

==================================================

ASSIGNMENT

===========

IDENTIFY THE VALID AND INVALID IDENTIFIERS AND MENTION THE REASON.

rateofInterest

rate@123

python

c#

c++

java

sbiloanrate

myClass

class\_1212

1\_class

=====================================================

IO OPERATIONS

==============

==> IO ==> INPUT AND OUTPUT OPERATIONS

INPUT ==> READING A VALUE

==> TAKING THE VALUE FROM THE KEYWORD.

==> input()

OUTPUT ==> WRITING A VALUE

==> DISPLAY ANYTHING ON THE SCREEN

print()

**Day6:** **IO Operations and Integer Datatype:**

IO OPERATIONS

==============

IO ==> INPUT AND OUTPUT

INPUT OPERATION:

================

TAKING OF THE DATA FROM THE KEYBOARD WHILE THE PROGRAM IS RUNNING

OR

READING OF THE DATA

pre-defined method: input()

Syntax for input()

==============

identifier = input()

or

identifier = input("Enter some text")

==> using the input(), we can allow to read any data.

OUTPUT OPERATION

==================

WRITING/DISPLAYING OF ANYTHING ON THE SCREEN

PRE-DEFINED METHOD: print()

Syntax for print()

==============

1) printing of single value

=====================

print(name of the value)

2) printing of more than one value using single print()

============================================

print(name1,name2,name3,...)

Note:

====

print() ==> by default reserve with new line.

3) printing of a value along with message

=================================

First value = 123

Second value = 234

123 is the first value

234 is the second value

The First value is : 123, The Second value is: 234

123 is the first value and 234 is the second value

print("message with single quote or double quote",name of the value)

print(name of the value,"message with single quotes or double quotes")

print("message",name of the value,"message",name of the value)

print(name of the value,"message",name of the value,"message")

4) printing of values along with messages with % symbol

==============================================

==> formatting of data with %

print("message %s"%(name of the value)) # for single value with message

print("message %s message %s message %s"%(name1,name2,name3)) # for more values along with messages

5) using format()

=============

print("Message {}".format(name of the data))

print("Message {} "message {}".format(name1,name2))

print("Message {dummy name1} message {dummy name2}".format(dummy name1 = name1,dummy name2 = name2))

print("message {3} message {1} message {2}".format(name1,name2,name3))

a = input() # we can't get any message screen to make confirm that why console is waiting

b = input("Enter something:") # the text or message make you confirm that why console is waiting

# printing single values

print(a) # first value

print(b) # second value

# printing of more than one values

print(a,b) # here printing of all values within the same line is possible

# PRINTING VALUES ALONG WITH MESSAGES

print("The Value of a = ",a)

print('The Value of b = ',b)

print(a,"is the value for a")

print(b,'is the value for b')

print("The Value of a = ",a,"and The Value of b = ",b)

print(a,": is the value of a ",b,': is the value of b')

# formatted output

print("The value of a is : %s"%(a))

print('The value of b is : %s'%(b))

print("The Value of a is: %s and the value of b is: %s"%(a,b))

# using format()

print("The Value of a = {}".format(a))

print('The Value of b = {}'.format(b))

print("The value of a = {} and the value of b = {}".format(a,b))

print("The value of a = {x} and the value of b = {y}".format(x = a,y = b))

print("The Value of a = {1} and the value of b = {0}".format(a,b))

=============================================================================

DATATYPES

==========

==> IN HOW MANY WAYS THE DATA CAN BE REPRESENTED IN PYTHON PROGRAM

==> ALL THE DATATYPES IN PYTHON ARE IN-BUILT/PRE-DEFINED.

BECAUSE FOR EVERY DATATYPE, THERE IS ONE PRE-DEFINED CLASS

==> CLASS ==> A COLLECTION OF DATA AND METHODS

==> TO ACCESS THE DATA OF ANY CLASS, WE NEED AN OBJECT.

EX: a = 10 ==> 10 in an integer class

# 10 is getting accessed using an object 'a'

==> DATATYPES ARE CLASSIFIED INTO:

1) PRIMITIVE DATATYPES/FUNDAMENTAL DATATYPES

2) COLLECTION DATATYPES

==> TO GET THE CLASS NAME OF ANY DATATYPE: WE CAN USE type()

1) PRIMITIVE DATATYPES/FUNDAMENTAL DATATYPES

==============================================

==> classified into:

1) numerical type ==> all the data with only numbers ==> ex: 123, 12.234, 12-23j etc.

2) non-numerical type ==> all the data without numbers ==> True, False

3) text based ==> the data with single quotes or double quotes ==> 'hi', "12345" ==> string data

1) numerical type

==============

integers

floats

complex

Note:

===

python is dynamically typed programming language

integers

======

binary ==> base-2 number ==> with only two literals (0 and 1) ==> 0b1100001010010

octal ==> base-8 number ==> with only 8 literals (0 to 7) ==> 0o71023, 0O1100010010

decimal ==> base-10 number ==> with 10 literals (0 to 9) ==> 12345, 10001010101 ==> for the decimals, no prefix is required

hexadecimal ==> base-16 number ==> with 16 literals (0 to 9 and alphabets: a to f) ==> 0x12013201, 0Xaf123

Note: any number, can be decoded by the Python Virtual Machine (PVM) (Interpreter) as decimal by default.

Note:

===

print() will always print any value in decimal format by default.

**Day07 : Integer Data Conversions:**

BASE CONVERSIONS

================

BINARY ==> BASE2 ==> WITH: 0 AND 1 ==> PREFIX ==> 0B ==> EX: 0B110011001

OCTAL ==> BASE8 ==> WITH: 0 TO 7 ==> PREFIX ==> 0O ==> EX: 0O112273

DECIMAL ==> BASE10 ==> WITH: 0 TO 9 ==> NO PREFIX ==> EX: 11223344

HEXADECIMAL ==> BASE16 ==> WITH: 0 TO 9 AND ALPHABETS(A TO F) ==> PREFIX: 0X ==> EX: 0XAF123

BINARY ==> DECIMAL

OCTAL ==> DECIMAL

HEXADECIMAL ==> DECIMAL

DECIMAL ==> BINARY

OCTAL ==> BINARY

HEXADECIMAL ==> BINARY

DECIMAL ==> OCTAL

BINARY ==> OCTAL

HEXADECIMAL ==> OCTAL

DECIMAL ===> HEXADECIMAL

BINARY ==>HEXADECIMAL

OCTAL ==> HEXADECIMAL

ANY NUMBER TO DECIMAL CONVERSION

================================

two methods:

1) print()

2) int() ==> used to convert any number into decimal

Syntax:

int(value)

a = 0b110011001 # binary

b = 0o123 # octal

c = 0xaf1 # hexadecimal

# decimal conversion using print()

print("The Binary in Decimal = ",a)

print("The Octal in Decimal = ",b)

print("The Hexadecimal in Decimal = ",c)

# decimal conversion using int()

print("The Binary in Decimal = ",int(a))

print("The Octal in Decimal = ",int(b))

print("The Hexadecimal in Decimal = ",int(c))

======================================================

BINARY CONVERSIONS

===================

==> pre-defined method: bin()

Syntax:

bin(value)

a = 22

b = 0o123

c = 0x1f2

print("The Decimal In Binary = ",bin(a))

print("The Octal in Binary = ",bin(b))

print("The Hexadecimal in Binary = ",bin(c))

=========================================================

OCTAL CONVERSIONS

=================

==> oct()

Syntax:

oct(value)

a = 22

b = 0b1100110011001

c = 0x1234

print("The Decimal in Octal = ",oct(a))

print("The Binary in Octal = ",oct(b))

print("The Hexadecimal in Octal = ",oct(c))

============================================================

HEXADECIMAL CONVERSION

=======================

hex()

Syntax:

hex(value)

a = 100

b = 0b110011001101

c = 0o123

print(hex(a))

print(hex(b))

print(hex(c))

================================================================

VARIABLE

========

==> NAMED MEMORY

USED TO STORE ANY VALUE.

==> CAN BE DEFINED IN TWO WAYS:

1) COMPILE TIME ==> DIRECT VALUE ASSIGNMENT

2) RUN TIME ==> input()

TYPECASTING:

===========

==> ALSO CALLED AS "TYPE CONVERSION"

==> USED TO CONVERT THE DATA FROM ONE FORM TO ANOTHER FORM.

Note:

====

input() ==> can able to read any value in string format by default.

2) WAP TO TAKE A BINARY AS AN INPUT AND PRINT ITS DECIMAL, OCTAL AND HEXADECIMAL.

NOTE:

====

RUN TIME VARIABLE DEFINITION

input() should be type casted.

Syntax:

to read a binary as an input using input()

int(input('text',2))

# WAP TO TAKE A Binary VALUE AS AN INPUT AND PRINT THAT VALUE IN Decimal, OCTAL AND HEXADECIMAL FORMATS.

a = int(input("Enter a binary number:"),2)

print(type(a))

print("To Decimal = ",int(a),a)

print("To Octal = ",oct(a))

print("To Hexadecimal = ",hex(a))

3) WAP TO TAKE AN OCTAL AS AN INPUT AND PRINT ITS BINARY, DECIMAL AND HEXADECIMAL.

Syntax:

int(input("text"),8)

# WAP TO TAKE AN OCTAL AS AN INPUT AND PRINT ITS BINARY, DECIMAL AND HEXADECIMAL.

num = int(input("Enter an octal value:"),8)

print("To Decimal = ",num,int(num))

print("To Binary = ",bin(num))

print("To Hexadecimal = ",hex(num))

4) WAP TO TAKE AN HEXADECIMAL AS AN INPUT AND PRINT ITS BINARY, OCTAL AND DECIMAL.

Syntax:

int(input("text"),16)

1) WAP TO TAKE A DECIMAL AND PRINT ITS BINARY, OCTAL AND HEXADECIMAL

a = int(input("Entre a decimal value:"))

print("To Binary = ",bin(a))

print("To Octal = ",oct(a))

print("To Hexadecimal = ",hex(a))

**Day8:** **Remaining Primitive Datatypes:**

DATATYPES:

=========

1) NUMBER TYPE:

==============

INTEGER DATATYPE ==> DECIMALS, BINARY, OCTAL AND HEXADECIMAL, BASE CONVERSIONS

FLOAT DATATYPE

COMPLEX DATATYPE

FLOAT DATATYPE

==============

==> NUMBER WITH DECIMAL POINT

EX: 0.001, 12.234 ETC.

==> PRE-DEFINED/INBUILT DATATYPE

PRE-DEFINED CLASS ==> float

Exponential Format:

===================

==> also called as "scientific form".

==> used to define too large values or too smaller values

ex: distance from moon to earth or sun to earth

12300000 ==> 123 X 10^+5 ==> 123/100 X 100 X 10^5 ==> 1.23 X 10^7 ==> 1.23e7

0.000000019 ==> 19 X 10^-7 ==> 19/100 X 100 X 10^-7 ==> 0.19 X 10^-5 ==> 0.19E-5

a = 0.0001

b = 12.345

print("The Type of a = ", type(a))

print("The Type of b = ", type(b))

"""

1) no binary data as float

Ex: 0b11001.1101 ==> invalid

2) no octal data as float

3) no hexadecimal data as float

==> we will get a syntax error.

"""

# c = 0b11001.110

# c = 0o12.2734

# c = 0x123.123

# print(type(c))

"""

1) to define the float data, we can allow to use only decimal literals (0 to 9).

2) float data can be defined in exponential format.

"""

c = 0.193 - 5

d = 123e7

print("The Type of c = ", type(c))

print("The Type of d = ", type(d))

print(a, b, c, d)

=================================================

979700000000000000 ==> 9797 X 10^+14 ==> 9797/10000 X 10000 X 10^14 => 0.9797 X 10^4 X 10^14 ==> 0.9797 X 10^18 ==> 0.9797E18

0.0000000000097 ==> 97 X 10^-11 ==> 97/100 X 100 X 10^-11 ==> 0.97 X 10^-9 ==> 0.97e-9

========================================================

COMPLEX DATATYPE:

=================

==> COMBINATION OF REAL PART AND IMAGINARY PART.

REAL DATA ==> WITH ANY INTEGER TYPE AND/OR FLOAT

IMAGINARY DATA ==> WITH EITHER DECIMAL OR FLOAT, BUT NOT WITH OTHERS LIKE: BINARY, OCTAL, HEXADECIMAL

SYNTAX:

real\_data +/- imagnary\_data

Note:

====

Imaginary data must be suffixed with 'j'

ex: a + bj, c - dj etc.

==> Pre-defined/Inbuilt datatype

pre-defined class: complex

a = 97 + 97j

b = 9.7 - 9.7j

# c = 0b11001 + 0b110j

c = 0b11001 - 9.7123j

print("Type of a = ", type(a))

print("Type of b = ", type(b))

print("Type of c = ", type(c))

=============================================================

2) non-numerical datatype:

=========================

==> the data without numerals

Boolean Datatype:

=================

==> can always be allowed to define with only two values:

True

False

Here: True and False ==> Keywords in python.

==> Pre-defined datatype:

class: bool

Note:

====

PVM (python virtual machine/interpreter) ==>

True ==> 1

False ===> 0

a = True

b = False

print("Type of a = ", type(a))

print("Type of b = ", type(b))

print("a = ", a)

print("b = ", b)

================================================

3) TEXT BASED DATATYPES:

========================

TEXT ==> STRING

==> THE COLLECTION/GROUP OF CHARACTERS

WHICH MUST BE ENCLOSED WITH SINGLE QUOTES OR DOUBLE QUOTES.

NOTE:

====

IN PYTHON, THERE IS NO CHARACTER DATA REPRESENTATION.

==> PRE-DEFINED DATATYPE:

CLASS: STR

a = 'c' # single quote with single character ==> string type

b = "V"

c = 'python'

d = "Python Programming"

print(type(a), type(b), type(c), type(d))

String Indexing:

================

==> to access the individual characters from the string.

==> the individual characters from the string can be allowed to access using indexing.

Syntax:

String\_Data\_Name[index number]

Here:

Index number ==> positive or negative

Positive ==> the string characters can be get allowed to access from left to right side.

Negative ==> the string characters can be get allowed to access from right to left side.

==> Range for Positive index values ==> 0 to total\_characters - 1

Ex: string with 10 characters ==> range for +eve index ==> 0 to 9

==> Range for Negative Index values ==> -1 to -total number of characters

Ex: string with 10 characters ==> range for -eve index ==> -1 to -10

Ex: str1 = "Python"

# Positive index ==> Left to right ==> Forward Accessing

str1[0] ==> P

str1[1] ==> y

str1[2] ==> t

str1[3] ==> h

str1[4] ==> o

str1[5] ==> n

str1[6] ==> exceed the range of index ==> an error

# Negative Index ==> Right to left ==> Reverse Accessing

str1[-1] ==> n

str1[-2] ==> o

str1[-3] ==> h

str1[-4] ==> t

str1[-5] ==> y

str1[-6] ==> P

# Indexing

str1 = "Python"

# Positive Indexing

print(str1[0])

print(str1[1])

print(str1[2])

print(str1[3])

print(str1[4])

print(str1[5])

# print(str1[6]) Index Error

print()

# Negative Indexing

print(str1[-1])

print(str1[-2])

print(str1[-3])

print(str1[-4])

print(str1[-5])

print(str1[-6])

Note:

====

1) Python strings can be allowed to define with any characters:

alphabets (upper case/lower case), digits, special characters.

Ex:"ravi@1122#" ==> correct

'ravi 27' ==> correct

2) No limit of length to define the strings.

String Slicing

==============

to access the specific part of the string

Ex: str1 = "Python Programming"

op1 ==> "Python"

op2 ==> "Programming"

op3 ==> "Ph oai"

Syntax:

string\_data\_name[start:stop:step]

Here:

start ==> from which index, the slicing of the string should start/begin

stop ==> until which index, the slicing need to continue

(before the value of stop)

step ==> the difference between the current index and next expected index.

Note:

====

1) step factor ==> not mandatory for all time.

If step factor ==> absent

default value ==> 1

2) stop is also absent

when we want to define the slicing until the last character.

3) start is also absent

when we want to define the string slicing from the first character

str1 = "Python Programming"

print(str1[0:6])

print(str1[:6])

print(str1[7:])

print(str1[0:10:2])

print(str1[::3])

print(str1[-1:-10:-1])

# Reverse of the string

print(str1[::-1])

**Day9: Collection Datatypes:**

id()

===

==> inbuilt method, which we can use to find the address location of the data (stack memory).

Syntax:

id(name of the data) or

id(value)

a = 100 #140733395174936

b = 200 # 140733395178136

c = 300 # 2954331794960

print(id(a))

print(id(b))

print(id(c))

# memory of the data is never be permanent,

# the memory of the data can be get assigned when the program will start and that assigned memory will be get released

# when the program is closed.

MUTABILITY Vs IMMUTABILITY

==========================

IMMUTABILITY

=============

ALL THE PRIMITIVE DATATYPES ARE ==> IMMUTABLE TYPE OF DATATYPES

IMMUTABILITY ==> WHEN WE HAVE DEFINED A DATA IN THE PROGRAM, WHICH IT CANNOT BE GET MODIFIED AGAIN AND AGIN.

IF THE MODIFICATION IS PERFORMED ON THE DATA, THE MEMORY WHICH WAS ASSIGNED EARLIER THAT MEMORY WILL BE GET CHANGED.

a = 100

b = 123.234

c = 12-23j

d = True

e = 'Python'

# before the modification

print(id(a),id(b),id(c),id(d),id(e))

print(a,b,c,d,e)

# a = 140733395174936

# b = 2061537095280

# c = 2061540524592

# d = 140733394379152

# e = 140733394059536

a = 1000

b = 0.0021e-27

c = 1222-2334j

d = False

e = 'abcde'

# after modification

print(id(a),id(b),id(c),id(d),id(e))

print(a,b,c,d,e)

MUTABILITY

==========

AFTER THE DEFINITION, THE DATA CAN BE ALLOWED FOR THE MODIFICATION.

AND THE MODIFICATION ON DATA IS POSSIBLE WITHIN THE SAME MEMORY ADDRESS.

EX: LIST, SETS, DICTIONARY ETC.

a = [1,3,5,7,9] # list

b = {11,12,13,14,15} # set

c = {'a' : 123,'b' : 121, 'c' : 222} # dictionary

# before the change

print(id(a))

print(id(b))

print(id(c))

print(a,b,c)

a[2] = 97

b.add(123)

c['d'] = 129779

print(id(a))

print(id(b))

print(id(c))

print(a,b,c)

===================================================================

COLLECTION DATATYPES

=======================

==> THE VARIABLE CAN BE DEFINED WITH MORE THAN ON E VALUE

==> THE DIFFERENT COLLECTION DATATYPES ARE:

1) SEQUENTIAL DATATYPES

==> LIKE STRINGS, THESE ARE ALSO INDEX BASED AND OREDRED.

==> THERE ARE TWO SEQUENTIAL DATATYPES:

1) LIST DATATYPE

2) TUPLE DATATYPE

2) NON SEQUENTIAL DATATYPES

==> NOT DEPENDING WITH INDEX BUT MAY OR MAY NOT BE ORDERED.

==> ARE CLASSIFIED INTO:

1) SET TYPE ==> {}

i) SETS

ii) FROZEN SETS

2) MAP TYPE ==> KEY AND VALUE PAIR FORMAT

i) DICTIONARY

3) BYTE TYPE ==> 0 TO 255

i) BYTES

ii) BYTEARRAY

1) LIST DATATYPE

================

==> LIST DATA CAN ALWAYS BE DEFINED WITH ==> []

==> LIST IS COLLECTION OF HOMOGENEOUS ELEMENTS AND HETEROGENEOUS ELEMENTS

HOMOGENEOUS ==> ALL VALUES OF LIST ==> SAME TYPE

HETERGENEOUS ==> THE DIFFERENT TYPES OF IETEMS ARE JOINED

==> THE LIST ELEMENTS IN [] MUST BE SEPARATED WITH COMMA

EX: [1,2,3,5]

==> LIST DATATYPE ==> INBUILT DATATYPE

CLASS ==> list

==> LIST IS ORDERED.

==> LIST IS INDEX BASED

WE CAN ALLOWED TO ACCESS THE INDIVIDUAL ELEMENTS OF A LIST WITH POSITIVE INDEX AND WITH NEGATIVE INDEX

WE CAN ALLOWED TO ACCESS THE PART OF THE LIST USING SLICING

==> LIST MUTABLE DATATYPE.

==> LIST CAN BE DUPLICATED.

# homogeneous list

ld1 = [1,3,5,7,9] # a list with decimals

ld2 = [0b11001,0o112233,0x112233, 1234] # list with integers

ld3 = [11.12,1.23e-2,22.334] # list with floats

ld4 = [11,12.23e-8,True,12-23j,'abcd'] # hetergeneous list

print(type(ld1),type(ld2),type(ld3),type(ld4))

print(ld1)

print(ld2)

print(ld3)

print(ld4)

print(ld1[0],ld1[1],ld1[2],ld1[3],ld1[4]) # positive indexing

print(ld1[-1],ld1[-2],ld1[-3],ld1[-4],ld1[-5]) # negative indexing

print(ld1[::-1])

print(ld2[::-1])

print(id(ld1))

ld1[3] = 97

print(ld1)

print(id(ld1))

===========================================

2) TUPLE DATATYPE

==================

==> the sequence of values can be represented within () with the comma separation ==> tuple data definition.

==> tuple can be homogeneous

==> tuple can be heterogeneous

==> tuple is ordered

==> tuple is index based

==> tuple can be duplicated.

==> tuple can be immutable.

a = (1,2,3,4,5,6,7) # homogeneous tuple

b = (11,0.0097, True, 12-23j,'abcd') # heterogeneous tuple

c = (1,1,3,5,7,9,1,3,3,5,7,9)

print("a = ",a)

print("b = ",b) # ordered.

print(a[0],a[1],a[2],a[3],a[4],a[5],a[6]) # positive indexing

print(a[-1],a[-2],a[-3],a[-4],a[-5],a[-6],a[-7]) # negative indexing

print(b[1:5:1])

print(b[::-1])

print(c)

print(id(c))

# c[0] = 9797

print(c)

print(id(c))

**Day10:**  **Non-Sequential Datatypes:**

DATATYPES

===========

NON-SEQUENTIAL DATATYPES

==========================

==> NOT INDEX BASED

==> MAY OR MAY NOT BE ORDERED.

==> CLASSIFIED INTO:

1) SET DATATYPES

2) MAP DATATYPES

3) BYTE DATATYPES

1) SET DATATYPES

================

==> THERE ARE TWO TYPES:

1) SETS

2) FROZENSETS

1) SETS

=======

==> ALWAYS BE DEFINED WITH {}

SET ELEMENTS/ITEMS IN {} MUST BE SEPARATED WITH COMMA

EX: {1,2,3,4,5,5}

==> PRE-DEFINED/INBUILT DATATYPE

PRE-DEFINED CLASS ==> set

FEATURES OF SET DATA:

=====================

1) SET IS HOMOGENEOUS.

2) SET IS HETEROGENEOUS.

3) SET CANNOT BE INCLUDE A LIST AS MEMBER.

NOTE:

if a set with list as a member ==> Type Error.

4) SET IS NOT AN ORDERED.

5) SET IS NOT INDEXED.

Note:

if an index is used to set for accessing individual elements ==> Type Error.

6) NO SLICING IS ALLOWED ON SETS.

7) MUTABLE

a = {1,2,3,4,5,6,7,8,9,10} # SET WITH INTEGERS => HOMOGENEOUS SET DATA

b = {111,0.0097, True, 12-23j, 'abcde',(1,2,3,4,5,6)} # set with different types ==> heterogeneous data

print(type(a))

print(type(b))

print(a)

print(b)

# print(a[0])

print(a)

print(id(a))

# to modify the set definition, the index is not supported

# with pre-defined method

# add()

a.add(100)

print(a)

print(id(a))

=================================================

FROZENSET

==========

==> similar to sets

==> is always be allowed to define with "frozenset()"

==> frozenset() ==> pre-defined method, which we can use to convert any collection to frozenset.

Syntax:

frozenset(collection data)

==> it is an inbuilt datatype

class ==> frozenset

Features - Frozenset

=================

1) Homogenous

2) Heterogeneous

3) Not Ordered

4) Not indexed

5) no slicing is possible

6) immutable.

a = "Python"

b = [1,3,5,7,9,11]

c = (2,4,6,8,10)

d = {1,2,3,4,5,6,7,8,9,11,13,15}

e = ['a',11,True, 12-23j,11.123]

print(type(a),type(b),type(c),type(d))

p = frozenset(a)

q = frozenset(b)

r = frozenset(c)

s = frozenset(d)

t = frozenset(e)

print(type(p),type(q),type(r),type(s),type(t))

print(p)

print(q)

print(r)

print(s)

print(t)

# print(p[0])

# p.add(100)

============================================================================

2) MAP DATATYPES

================

==> can always be defined with key and value pair format.

key : value

==> There is only one map datatype:

Dictionary

Dictionary Datatype:

===============-

==> can be defined with {},

here the elements are in the "key-value" pair format

key : value

==> the key-value pair is an element/item/member of the dictionary

==> these members must be placed within {} using the comma separation.

==> Inbuilt datatype

pre-defined class: dict

Features of Dictionary:

===================

about keys:

=========

1) keys are with any datatype

2) keys must be unique

about values:

==========

1|) values are with any datatype

2) values not to be unique.

1) Homogenous

2) Heterogeneous

3) Is an Ordered.

4) Not an index supported.

Note:

if an index is used to access the dictionary elements ==> Key Error.

5) Dictionary values can be accessed with its corresponding keys.

Syntax:

dictionary\_name[key name]

6) Mutable

a = {'key1' : 111,'key2' : 222,'key3' : 333}

b = {'a' : 11,True : 11.123, 1-2j : False}

print(type(a))

print(type(b))

print(a)

print(b)

# print(a[0])

print(a['key2'])

print(b[True])

print(id(a))

a['key2'] = 2222

print(a)

print(id(a))

==================================================

3) Byte Type

===========

==> there are two types:

1) bytes

2) bytearray

Note:

====

1 byte ==> 8-bits

2^8 ==> 256 characters

0 to 255

==> bytes and bytearray data can be always allowed to define with values from 0 to 255 only.

1) bytes

=======

==> can always be defined with bytes()

==> bytes() can allow any collection to convert into bytes.

Syntax:

bytes(collection data)

==> Pre-defined datatype

Predefined class ==> bytes

Features of Bytes Data:

===================

1) Homogeneous only.

2) Ordered.

3) Index based.

4) Slicing is allowed.

5) Immutable.

# a = 'abcdef'

b = [11,13,15,17,19]

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

d = {1,3,5,7,9}

e = {11:10,21:11,31:12,41:13,51:14,61:15}

print(type(b),type(c),type(d),type(e))

# p = btyes(a)

q = bytes(b)

r = bytes(c)

s = bytes(d)

t = bytes(e)

print(type(q),type(r),type(s),type(t))

print(q)

print(q[0])

print(q[-1])

print(q[::-1])

print(r[::-2])

print(id(q))

# q[0] = 101

=====================================

2) bytearray

==========

a = bytearray([10,20,30,40,50,60])

b = bytearray((11,21,31,41,51,30))

print(type(a),type(b))

print(a)

print(b)

print(a[0])

print(b[::-1])

print(id(a))

a[0] = 100

print(a)

print(id(a))

**Day11: Type Casting:**

NONE DATATYPE

==============

None ==> keyword in python

==> when we want to define a variable with nothing

Syntax:

name of the data = None

a = None

print(id(a))

print(a)

===================================

TYPE CASTING

=============

==> ALSO CALLED AS "TYPE CONVERSION"

==> WHEN WE WANT TO CONVERT A DATA FROM ONE FORM TO ANOTHER FORM ==> TYPE CONVERSION IS USED.

==> THE INBUILT METHODS FOR TYPE CASTING ARE:

int()

float()

complex()

bool()

str()

int()

===

==> when we need to convert any data/value into decimal, int() is used.

Syntax:

int(the value)

Rules:

=====

1) any base value of an integer, can be converted into decimal.

2) any float value, can be allowed for the decimal conversion

Ex: int(0.0097) ==> 0

int(0.98e-2)

3) No complex value can be converted into decimal.

Note: int(complex) ==> Type Error

4) Boolean values (True, False) are considered for the decimal conversion.

5) Not all the string objects are allowed for decimal conversions.

If string with numerals (integer(decimal)) ==> '1234' ==> possible for decimal conversion

If string with non-numerals ==> not possible

Ex: "12.23", "abcd" ==> not allowed. ==> Value Error.

a = 0b11001

b = 0o112276

c = 0x1122ff

d = 0.00879

e = 1.23e-7

# f = 12-23j

f = True

g = False

h = '1234'

# i = '12.32'

i = 'abcd'

print("To Decimal Conversion from any base value : ")

print(int(a))

print(int(b))

print(int(c))

print("To Decimal Conversion from any Float Number:")

print(int(d))

print(int(e))

# print(int(f))

print("The Decimal Conversion from Boolean:")

print(int(f))

print(int(g))

print("String to Decimals:")

print(int(h))

# print(int(i))

========================================

float()

====

==> used to convert any data/value to float data

Syntax:

float(data)

Rules:

=====

1) Any Integer data (includes: binary, octal, hexadecimal and decimal) allowed for float conversion.

2) No complex value is allowed for the float conversion. ==> Type Error

3) Boolean values are possible to float conversion.

4) when a string with any decimal and with any float is allowed for the float conversion

If string with alphabets and other special characters ==> Value Error

# Any Integer to Float Conversion

print(float(9876))

print(float(0b110001)) # binary ==> decimal ==> float

print(float(0o1122)) # octal ==> decimal ==> float

print(float(0x1122)) # hexadecimal ==> decimal ==> float

# complex to float conversion

# print(float(10+20j))

# Boolean to float conversion

print(float(True)) # boolean ==> decimal ==> float

print(float(False))

# string to float conversion

print(float('123'))

print(float('111.123'))

# print(float('0b11001'))

======================================================

complex()

========

==> used to convert any data to complex number

Syntax:

======

1) with single parameter

complex(value)

Here:

(+/-)value is the parameter ==> is understood as "real data" by default

Imaginary ==> 0j

ex: complex(10) ==> 10 + 0j

2) with two parameters

complex(val1,val2)

Here:

val1 ==> real value ==> +eve/-eve

val2 ==> imaginary value ==> +eve/-eve

op = val1 +/- val2j

Rules:

====

1) any base value of an integer is considered as real data and imaginary data also.

2) float ==> complex

3) boolean ==> complex

4) string with decimal and string with floats ==> complex

Note:

====

if complex(par1,par2):

par1 ==> string

par2 ==> not accepted.

# a = 23-0b110j

print(complex(0b1101))

print(complex(0b11001,0x1122))

print(complex(11.223))

print(complex(11.22,-22.34))

print(complex(True))

print(complex(True,False))

print(complex(False))

print(complex('1122'))

# print(complex('1122',9.123))

print(complex('0.997789'))

# print(complex('0b11001'))

===========================================

bool()

=====

==> used to convert any data into boolean

Syntax:

bool(any value)

print(bool(0))

print(bool(1))

print(bool(10))

print(bool(-10))

print(bool(0.0))

print(bool(0.000001))

print(bool(0+0j))

print(bool(0-0j))

print(bool(1-0j))

print(bool(''))

print(bool(' '))

print(bool('a'))

=====================================

5) str()

=====

==> used to convert any data into string data.

Syntax:

str(any value)

print(str(0b11001))

print(str(0o1122))

print(str(0x1122))

print(str(112233))

print(str(12e-23))

print(str(11.00043))

print(str(11-23j))

print(str(True))

print(str(False))

**DAY12: Operators Part\_01 :**

FORMAT SPECIFIER

==> SYMBOL

MACHINES CAN READ THE TYPE OF THE DATA.

==> %

INTEGER ==> %d

FLOAT ==> %f

CHARACTER ==> %c

% ==> make print the data in the specified format

%s ==> %(a,b,c)

true, false ==> java

true ==> never be 1

false ==> never be 0

True, False ==> Python

if, else

True ==> 1 (int)

False

OPERATORS

===========

statement

========

==> a line of python code

==> with or without ending semi-colon

program:

=======

==> group of statements

expression

========

==> the joining of operand and operator

Ex: 123 \* 2

a = int(input())

block

====

==> group of one or more than one statement ==> block

In java:

block of code{

int a = 100;

float f = 21.23f;

}

In Python:

block of code ==> indentation

print("Hello")

a = 97

b = 79

if a > b:

big = a

print("Biggest number = ",big)

else:

big = b

print("Biggest number = ",big)

operand

======

a = int(input("Enter an Integer:")) 97

b = int(input("Enter an Integer:")) 79

print("The Sum is = ", a + b)

a + b ==> 97 + 79 ==> 176

==> a data on which we can define an operation called as "operand"

==> an operand is a data/variable

operator

======

==> a symbol

which is used to specify the type of operation.

sum ==> +

subtraction ==> -

multiplication ==> \*

ex: a+b ==> a and b ==> operands

+ ==> operator

types of operators

==============

==> categorized into three types:

1) Unary operators ==> always be defined with single operand

ex: logical not operator, bitwise complement, unary minus etc.

==> not a ==> logical not

==> ~a ==> bitwise complement

==> -a ==> unary minus

2) Binary Operators ==> always be used with two operands

ex: 97 + 79, 9 / 2, 97 > 79 etc.

3) Ternary Operators ==> used with three operands

also called as "conditional operator"

Syntax:

result = part-1 if condition else part-2

ex:

a = 100

b = 200

c = 300

big = a if a > b and a > c b elif b > a and b > c else c

==> big = 100 > 200 else 200

==> big = 200

==> based on the number of operands

Operator classification

=================

==> based on the type of the operation:

the operators are classified into:

1) Arithmetic operators

2) Assignment operator

3) Compound Operator

4) Relational Operators

5) Logical Operators

6) Bitwise Operators

7) Conditional Operators

8) Special Operators

1) Arithmetic operators

=================

Unary Minus ==> - ==> to change the sign of the data.

plus ==> + ==> addition

minus ==> - ==> subtraction

asterisk ==> \* ==> multiplication

three division operators:

normal division ==> backslash ==> / ==> return quotient in float

floor division ==> double backslash ==> // ==> return quotient int

modulo division ==> percentage ==> % ==> return a remainder

Power Operator ==> double asterisk ==> \*\*

Ex: 2^3 ==> 2 \*\* 3 ==> 8

print(10 + 20)

print(10 - 20)

print(10 \* 20)

print(10/20)

print(10//20)

print(10 % 20)

print(10 \*\* 20)

a = -10

print(a)

print(-a)

==============================

2) Assignment operator

==================

==> =

==> used to assign a value to the expression/variable.

Ex: a+b = 10

a = 100

Syntax:

expression/variable = value

3) Compound Operator

==================

==> when we can join other operators with assignment operator ==> compound operator

Ex: +=, -=, \*=, /=,//=,%=, \*\*=

&=,|= etc.

a = 1

a = a + 10 ==> 11

a += 10 ==> 1 + 10 ==> 11

a = 10

print(a)

# a = a + 10

a += 10

print(a)

# a = a - 5

a -= 5

print(a)

# a = a \* 2

a \*= 2

print(a)

# a = a & 2

a &= 2

print(a)

**Day13: Operators Part\_02:**

PYTHON OPERATORS

===================

4) RELATIONAL OPERATORS

========================

==> ALSO CALLED AS "COMPARISION OPERATORS"

==> RETURNS: BOOLEAN VALUES (True/False)

==> THE RELATIONAL OPERATORS ARE:

< ==> LESS THAN

> ==> GREATER THAN

<= ==> LESS THAN OR EQUALS TO

>= ==> GREATER THA OR EQUALS TO

== ==> EQUALS TO

!= ==> NOT EQUALS TO

==> ARE BINARY OPERATORS

a = int(input("Enter some value:"))

b = int(input("Enter some value:"))

print(a < b) # a is less than b

print(a > b) # a is greater than b

print(a <= b) # a is either less or equal to b

print(a >= b) # a is either greater or equals to b

print(a == b) # a is equals to b

print(a != b) # a is not equals to b

DIFFERENCE BETWEEN = AND ==

============================

== ==> CHECK TWO NUMBERS ARE EQUAL OR NOT

EX:

A == 10;B= 10

A == B==> TRUE

A = B # B VALUE WILL BE ASSIGNED TO A

# equality

# a == 100 Name Error

a = 100

b = 100

print(a == b)

# print(a = b) # Type Error

========================================

5) LOGICAL OPERATORS

=====================

==> ARE ALLOWED TO DEFINE ON LOGICAL VALUES (BOOLEAN VALUES)

==> PYTHON CAN BE ALLOWED TO USE THE LOGICAL OPERATORS WITH ANY OTHER DATA.

==> THREE LOGICAL OPERATORS:

1) LOGICAL AND ==> and

2) LOGICAL OR ==> or

3) LOGICAL NOT ==> not

==> and/or operators ==> binary operators

==> not ==> unary operator

==> all these logical operators, working based on the truth tables.

and operator

==========

a b a and b

======================

False False False

False True False

True False False

True True True

Note: (Other than Booleans)

=====================

1) if both inputs are non-zero values

output ==> second input value(second operand)

2) if first operand ==> zero

and second operand ==> non-zero

output ==> zero

3) if first operand ==> non-zero

and second operand ==> zero

output ==> zero

print(False and False)

print(False and True)

print(True and False)

print(True and True)

# define with integers

print(10 and 20)

print(0b11001 and 0o1122)

print(0 and 10)

print(20 and 0)

print(-10 and -20)

# with floats

print(1.2 and 2.3)

print(0.0 and 2.3)

print(1.2 and 0.0)

# WITH COMPLEX

print(1-2j and 1+2j)

# with strings

print('a' and 'b')

print('' and 'b')

print(True)

print('a' and '')

print('Ravi')

===================================

or operator

=========

a b a or b

=====================

False False False

False True True

True False True

True True True

Note: (defining with other than boolean values)

====================================

1) if both operands ==> non-zero

output ==> First operand

2) if first operand ==> zero

second operand ==> non-zero

output ==> Second operand

3) if the first operand ==> non-zero

second-operand ==> zero

output ==> first operand

print(False or False)

print(False or True)

print(True or False)

print(True or True)

# with integers

print(10 or 20)

print(-10 or -20)

print(-10 or 20)

print(0 or 20)

print(10 or 0)

print(0 or 0)

# with floats

print(10.2 or 20.3)

print(0.0 or 20.3)

print(10.2 or 0.0)

# with complex

print(10-2j or 10+3j)

# with string

print('a' or 'b')

print('' or 'b')

print('a' or '')

===================================

not operator

=========

a not a

============

True False

False True

Note:

====

can accept any type of input

output ==> boolean

print(not True)

print(not False)

# with integers

print(not 10)

# with floats

print(not 0.0001)

# with complex

print(not 1-2j)

# with string

print(not 'a')

print(not '')

==================================

6) TERNARY OPERATOR

====================

==> ALSO CALLED AS "CONDITIONAL OPERATOR".

Syntax:

value3/operand3 = value1/operand1 if test-condition else value2/operand2

# WAP IN PYTHON USING TERNARY OPERATOR TO CHECK WHETHER THE GIVEN INTEGER IS POSITIVE OR NEGATIVE.

number = int(input("Enter a value:"))

# positive ==>? number > 0

# negative ==> number < 0

# result = "number is positive" if number > 0 else "number is negative"

#

# print(result)

print(number,"is positive number") if number > 0 else print(number,"is negative number")

===========================================

# WAP IN PYTHON USING TERNARY OPERATOR TO FIND WHETHER THE GIVEN NUMBER IS EVEN NUMBER OR ODD NUMBER.

number = int(input("Enter a value:"))

# even ==> number / 2, remainder == 0

# odd ==> number / 2, remainder != 0

print(number,"is an even number.") if number % 2 == 0 else print(number,"is an odd number.")

======================================================

# WAP IN PYTHON TO FIND THE MAXIMUM AMONG THREE NUMBERS USING CONDITIONAL OPERATOR.

a = int(input("Enter a value:")) # 7

b = int(input("Enter a value:")) # 1

c = int(input("Enter a value:")) # 9

maximum = a if (a > b and a > c) else (b if b > c else c)

print("The Maximum value = ",maximum)

SPECIAL OPERATORS

==================

==> TWO SPECIAL OPERATORS:

1) MEMBERSHIP OPERATORS ==> in and not in

======================================

on collection data and on strings

used to check whether the specified element is belonging to the collection or not.

return: Boolean Values

a = 'python'

b = [1,3,5,7,9]

print('P' in a) # False

print('P' not in a) # True

print(9 in b) # True

print(9 not in b) # False

2) IDENTITY OPERATORS ==> is and is not

===================================

==> used to check the address of the data

whether the two objects are belonging to same address or not.

a = 100

b = 200

c = 100

print(id(a))

print(id(b))

print(id(c))

print(a is c) # True

print(a is not b) # True

**Day14: Operators Part\_03:**

BITWISE OPERATORS

==================

==> ARE ALWAYS BE DEFINED WITH INTEGERS AND BOOLEAN ONLY

==> BITWISE OPERATORS ==> DATA

BY CONVERTING DATA INTO BINARY (BITS)

==> THE BITWISE OPERATORS ARE:

1) BITWISE AND ==> & ==> BINARY

2) BITWISE OR ==> | ==> BINARY

3) BITWSIE XOR (EXCLUSIVE OR) ==> ^ ==> BINARY

4) BITWISE COMPLEMENT ==> ~ ==> UNARY ==> 2's complement

5) LEFT SHIFT OPERATOR ==> << ==> BINARY

6) RIGHT SHIFT OPERATOR ==> >> ==> BINARY

==> LIKE LOGICAL OPERATORS, SOME OF BITWISE OPERATORS ALSO ALLOWED TO DEFINE WITH TRUTH TABLES.

Note:

====

==> IN JAVA,

UNSIGNED RIGHT SHIFT OPERATION (>>>) ==> FOR ONLY +EVE NUMBER

==> IN PYTHON, UNSIGNED RIGHT SHIFT IS NOT SUPPORTED.

1) BITWISE AND ==> &

==================

a b a & b

====================

0b0 0b0 0b0

0b0 0b1 0b0

0b1 0b0 0b0

0b1 0b1 0b1

print(0b0 & 0b0)

print(0b0 & 0b1)

print(0b1 & 0b0)

print(0b1 & 0b1)

# print() always able to print the number in decimal format only

# with decimal

print(10 & 20)

print(0 & 10)

print(10 & 0)

print(-1 & 30)

print(-100 & 20)

print(100 & -1)

print(30 & -2)

2) BITWISE COMPLEMNT

====================

==> USED TO FIND THE 2'S COMPLEMENT OF THE NUMBER

+ ==> -

- ==> +

Note:

====

2's complement ==> 1's complement + 1

1's complement ==> 0 ==> 1 & 1 ==> 0

1) while calculating 2's complement,

if there is no carry at the last bit of the data:

==> we should calculate 2's complement for the result again

when we have performed the 2's complement for twice,

add -1 to above result.

2) when there is carry while finding the 2's complement,

add that carry to the last bit (right side).

print(~10)

print(~0)

print(~-10)

=============================================

3) BITWISE OR OPERATOR

======================

a b a | b

===================

0b0 0b0 0b0

0b0 0b1 0b1

0b1 0b0 0b1

0b1 0b1 0b1

print(0b0 | 0b0)

print(0b0 | 0b1)

print(0b1 | 0b0)

print(0b1 | 0b1)

print(10 | 20)

print(True | False)

# print(12.2 & 23.3) # Type Error

# print(12.2 | 23.3)

============================================

4) Bitwise Xor

===========

a b a ^ b

=====================

0 0 0

0 1 1

1 0 1

1 1 0

print(0 ^ 0)

print(0 ^ 1)

print(1 ^ 0)

print(1 ^ 1)

print(10 ^ 20)

==========================================

LEFT SHIFT OPERATOR

=====================

Syntax:

data << n.times

Formula:

=======

data \* 2 ^ n.times

Note:

====

left shift is always double the value

RIGHT SHIFT OPERATOR

==========================

Syntax:

data >> n.times

Formula:

data // 2^n.times

Note:

====

because of the right shift the data should be half

print(9 << 1)

print(9 << 2)

print(9 << 3)

print(9 << 4)

print(9 >> 1)

print(9 >> 2)

print(9 >> 3)

print(9 >> 4)

============================

Operator Precedence & Associativity

============================

precedence ==> the order/rank

associativity ==> direction

a \*\* b + 2 \* 23 \*\* 4

1 - 2 \* 3 / 4

**DAY15: Conditional Statements:**

CONTROL STATEMENTS

====================

for:

1) to execute only the selected part of the program only

2) to make execute the selected part of the program/block of program repeatedly

3) during the execution of the program, to stop execution uncertainly

==> three types:

1) conditional statements/selection statements

2) loop statements/iterative statements

3) loop control statements/jump statements

1) conditional statements/selection statements

====================================

==> three types:

1) if statement

2) if-else statements

3) if-elif-else statements

==> keywords: if, else, elif

Note:

====

switch statement ==> not supposed to use.

1) if statement

===========

Syntax:

if test\_condition:

statement-1

statement-2

statement-3

test-condition:

can be allowed to define with relational operators

boolean values

# WAP TO CHANGE THE SIGN OF THE NUMBER.

number = int(input("Enter a value:")) # -7

if number < 0:

number = -number

print("The number after sign change = ", number)

print("The number = ", number)

================================================

2) if-else statements

================

Syntax:

if test\_condition:

statement-1

statement-2

statement-3

else:

statement-4

statement-5

statement-6

# WAP TO FIND WHETHER THE GIVEN NUMBER IS POSITIVE OR NEGATIVE.

number = int(input("Enter a value:"))

if number > 0:

print(number, "is a positive number.")

else:

print(number, "is a negative number.")

=====================================

# WAP TO CHECK WHETHER THE GIVEN FOUR INTEGERS WILL FORM A SQUARE OR NOT.

side1 = int(input("Enter an integer:"))

side2 = int(input("Enter an integer:"))

side3 = int(input("Enter an integer:"))

side4 = int(input("Enter an integer:"))

# if side1 == side2 and side1 == side3 and side1 == side4:

if side1 == side2 == side3 == side4:

print("The given four side value will form a square.")

else:

print("The given four side values will not form a square.")

=================================================

ASSIGNMENTS:

=============

USING IF-ELSE:

USING CONDITIONAL OPERATORS

1) WAP IN PYTHON TO CHECK WHETHER THE GIVEN NUMBER IS EVEN OR ODD NUMBER.

2) WAP IN PYTHON TO FIND THE BIGGEST NUMBER AMONG TWO NUMBERS.

3) WAP TO FIND THE SUM OF TWO NUMBERS WHEN FIRST NUMBER IS SMALLER THAN SECOND OTHERWISE FIND THE DIFFERENCE.

==================================================

3) if-elif-else statements

==================

Syntax:

if test\_condition1:

statement-1

statement-2

elif test\_condition2:

statement-3

statement-4

elif test\_condition3:

statement-5

statement-6

else:

statement-7

statement-8

# WAP TO CHECK THE TYPE OF TRIANGLE.

"""

Equilateral ==> side1 == side2 == side3

Isosceles ==> side1 == side2 or side2 == side3 or side1 == side3

Scalene ==> side!= side2 != side3

"""

side1 = int(input("Enter a side:"))

side2 = int(input("Enter a side:"))

side3 = int(input("Enter a side:"))

if side1 == side2 == side3:

print("This is Equilateral Triangle.")

elif side1 == side2 or side1 == side3 or side2 == side3:

print("This is Isosceles Triangle.")

else:

print("This is Scalene Triangle.")

===================================

# WAP TO FIND THE SMALLEST NUMBER AMONG 5 INTEGERS.

a = int(input("Enter a value:"))

b = int(input("Enter a value:"))

c = int(input("Enter a value:"))

d = int(input("Enter a value:"))

e = int(input("Enter a value:"))

if a < b and a < c and a < d and a < e:

smaller = a

elif b < c and b < d and b < e:

smaller = b

elif c < d and c < e:

smaller = c

elif d < e:

smaller = d

else:

smaller = e

print("The Smallest number = ", smaller)

# WAP TO FIND THE GRADE OF THE STUDENT BY TAKING 5 SUBJECT MARKS

"""

PERCENTAGE >= 85 ==> A+

PERCENTAGE < 85 AND PERCENTAGE >= 75 ==> A

PERCENTAGE < 75 AND PERCENTAGE >= 65 ==> B

PERCENTAGE < 65 AND PERCENTAGE >= 55 ==> C

PERCENTAGE < 55 AND PERCENTAGE >= 40 ==> D

PERCENTAGE < 40 AND PERCENTAGE >= 33 ==> E

PERCENTAGE < 33 ==> FAIL

"""

s1 = int(input("Enter a subject marks:"))

s2 = int(input("Enter a subject marks:"))

s3 = int(input("Enter a subject marks:"))

s4 = int(input("Enter a subject marks:"))

s5 = int(input("Enter a subject marks:"))

total\_marks = s1 + s2 + s3 + s4 + s5

percentage = total\_marks // 5

# floor division ==> //

# when we need a quotient in integer format ==> floor division

# normal division ==> /

# when we need a quotient in float format ==> normal division

# modulo division ==> %

# to get a remainder as an output ==> modulo division

print("The Total score of the student = ", total\_marks)

print("The Percentage of the student = ", percentage)

if percentage >= 85:

grade = 'A+'

elif 75 <= percentage < 85:

grade = 'A'

elif 65 <= percentage < 75:

grade = 'B'

elif 55 <= percentage < 65:

grade = 'C'

elif 40 <= percentage < 55:

grade = 'D'

elif 33 <= percentage < 40:

grade = 'E'

else:

grade = 'Fail'

print("The Grade of the student = ", grade)

**DAY16:** **Loop Statements:**

LOOP STATEMANTS

=================

==> ALSO CALLED AS "ITERATIVE STATEMENTS"

EX: PRINTING NUMBERS FROM 1 TO 10.

i = 1

print(i) # 1

i = i + 1

print(i) # 2

i = i + 1

print(i) # 3

i = i + 1

print(i) # 4

i = i + 1

print(i) # 5

i = i + 1

print(i) # 6

i = i + 1

print(i) # 7

i = i + 1

print(i) # 8

i = i + 1

print(i) # 9

i = i + 1

print(i) # 10

==> to define loops:

three requirements:

1) initialization

2) condition

3) update

==> two types of loop statements:

1) while loop

2) for loop

Note:

====

like other high-level programming languages,

python does not support "do-while".

1) while loop

==========

Syntax:

=====

initialization

while condition:

statement-1

statement-2

update

outside loop statements

# WAP IN PYTHON TO PRINT NUMBERS FROM 1 TO 10 USING WHILE LOOP.

i = 1 # initialization

# i is called as "loop variable/iteration variable"

while i <= 10: # condition

print(i)

i = i + 1 # update

============================

# WAP IN PYTHON TO PRINT NUMBERS FROM 1 TO 10 USING WHILE LOOP.

i = 1 # initialization

# i is called as "loop variable/iteration variable"

while i <= 10: # condition

print(i,end = '\t')

# i = i + 1 # update

i += 1

=============================

# WAP IN PYTHON TO PRINT THE NUMBERS FROM 10 TO 1 USING WHILE LOOP.

i = 10 # initialization

while i >= 1: # condition

print(i,end = '\t')

i -= 1 # update

========================

# WAP TO PRINT THE MULTIPLICATION TABLE OF THE GIVEN INTEGER.

num = int(input("Enter an integer:"))

i = 1 # initialization

while i <= 10: # condition

print(num,"X",i,"=",num \* i)

i += 1 # update

======================================

# WAP IN PYTHON TO PRINT THE MULTIPLE OF THE GIVEN INTEGER.

# 7, 14, 21, 28,...

num = int(input("Enter an integer:"))

i = 1

while i <= 100:

if i % 7 == 0:

print(i,end = '\t')

i += 1

==========================

# WAP TO FIND THE SUM OF INDIVIDUAL DIGITS OF THE GIVEN NUMBER.

"""

9876 ==> 9 + 8 + 7 + 6 ==> 30

9876 ==> 9 x 10^3 + 8 x 10^2 + 7 x 10^1 + 6 x 10^0

9876, DIVIDE WITH 10

QUOTIENT ==> 987 98 9 0

REMAINDER ==> 6 7 8 9

SUM OF REMAINDER

"""

number = int(input("Enter an integer:")) # 9876

n = number # n = 9876

sum\_dig = 0

while n != 0:

ind\_dig = n % 10 # 9%10 ==> 6 7 8 9

sum\_dig += ind\_dig # 30

n //= 10 # 987 98 9 0

print("The Sum of Individual Digits of",number,"is = ",sum\_dig)

==================================================

# WAP IN PYTHON TO COUNT THE NUMBER OF DIGITS OF THE GIVEN NUMBER.

num = int(input("Enter an integer:"))

n = num

cnt\_dig = 0

while n > 0:

n //= 10

cnt\_dig += 1

print("The Number of Digits in ", num, "is = ", cnt\_dig)

**DAY17:** **while Loop practice:**

WHILE LOOP:

===========

# WAP TO COUNT THE NUMBER OF DIGITS IN A GIVEN NUMBER.

# for the counting of digits in number, no method is available.

# convert a number ==> string

# find the length of the string ==> count digits of number

num = int(input("Enter a value:"))

num\_str = str(num) # convert a number into a string

print(type(num))

print(type(num\_str))

"""

len() ==> pre-defined method

used to find the number of characters in a given string.

Syntax:

len|(str\_data)

"""

print("The number of digits of",num,"is = ",len(num\_str))

===================================

# WAP IN PYTHON TO CHECK WHETHER THE GIVEN NUMBER IS PALINDROME NUMBER OR NOT.

"""

NUMBER

FIND REVERSE

REVERSE = 0

REVERSE = REVERSE \* 10 + IND\_DIG

REVERSE == NUMBER ==> PALINDROME NUMBER

1 x 10 = 10

10 x 10 = 100

100 x 10 = 1000

"""

num = int(input("Enter a decimal value:")) # 1221

n = num # n = 1221

rev\_num = 0

while n != 0:

ind\_dig = n % 10 # 1 2 2 1

rev\_num = rev\_num \* 10 + ind\_dig # 1221

n //= 10 # 122//10 ==> 12//10 ==> 1//10 ==> 0

if rev\_num == num:

print(num,"is a palindrome number.")

else:

print(num,"is not a palindrome number.")

==========================

# WAP TO FIND WHETHER A GIVEN NUMBER IS ARMSTRONG NUMBER OR NOT.

"""

3-DIGIT: XYZ

X^3 + Y^3 + Z^3 == XYZ ==> ARMSTRONG NUMBER

4-DIGIT: ABCD

A^4 + B^4 + C^4 + D^4 == ABCD

"""

num = int(input("Enter a number:")) # 1234

cnt\_dig = len(str(num)) # 4

n = num # n = 1234

s\_dig = 0

while n > 0:

ind\_dig = n % 10 # 4 3 2 1

powers = ind\_dig \*\* cnt\_dig # 64 81 16 1

s\_dig += powers # 64 + 81 + 16 + 1

n //= 10 # 123//10 == 12//10 == 1//10 == 0

if s\_dig == num:

print(num, "is an Armstrong number.")

else:

print(num, "is not an Armstrong number.")

**Day18: Loop Statements Part\_02:**

LOOP STATEMENTS

================

FOR LOOP

=========

==> for ==> keyword

==> when we want execute a block of code repeatedly with each element from the collection

Syntax:

for iteration\_variable in collection:

block of code

ex: for i in [1,2,3,4]:

print(i) # 1 2 3 4

# ITERATING ON STRING DATA USING FOR LOOP.

str\_data = input("Enter a string data:")

# string at an index of 0 ==>

# string at an index of 1 ==>

index = 0

for i in str\_data:

# print(i)

# print("The Character at positive index of",index,"is = ",i)

print("The character at negative index of",index-len(str\_data),"is = ",i)

index = index + 1

# "Python"

# positive index ==> 0 to 5

# negative index ==> -1 to -6

# P ==> 0 -6 ==> 0 - 6 ==> -6

index = -1

for j in str\_data[::-1]:

print("The Character at ",index,"is = ",j)

index -= 1

======================================

# WAP IN PYTHON TO FIND THE PRODUCT OF ALL ODD NUMBERS FROM THE GIVEN LIST

list\_data = eval(input("Enter a list data:"))

result = 1

for i in list\_data:

if i % 2 != 0:

result = result \* i

print("The Product of all Odd numbers = ",result)

====================================

ASSIGNMENT:

===========

1) WAP TO TAKE A TUPLE DATA FROM THE KEYBOARD AND PRINT THE TUPLE ELEMENTS ALONG WITH POSITIVE INDEX AND ALSO WITH NEGATIVE INDEX.

HINT:

====

THE ELEMENT AT INDEX POSITIVE 0 AND NEGATIVE INDEX -7 = 100

2) WRITE A PYTHON TO ACCEPT A LIST AND PRINT ITS SUM OF ALL INDIVIDUAL ELEMENTS USING FOR LOOP.

===================================

==> FOR LOOP CAN ALSO BE ALLOWED TO MAKE EXECUTE THE BLOCK OF CODE ON EACH ELEMENT FROM THE RANGE OF VALUES.

range()

=====

==> used to generate a range of values

Syntax:

range(value) ==> generate a values from '0' to 'value-1'

range(start,stop) ==> generate values from 'start' to 'stop-1'

range(start,stop,step) ==> generate values from 'start' to 'stop-1' with the difference of 'step'

for loop syntax with range() is:

for iteration\_variable in range():

block of code

for i in range(10):

print(i,end = "\t")

print()

for j in range(10,20):

print(j,end = "\t")

print()

for k in range(10,100,20):

print(k,end = "\t")

print()

for p in range(100,1,-10):

print(p,end = "\t")

Nested Loops

===========

writing of a loop block in another loop block

# WAP TO PRINT ALL ARMSTRONG NUMBERS FROM 1000 TO 5000.

"""

ARMSTRONG NUMBERS

THE SUM OF nth POWERS OF INDIVIDUAL DIGITS OF A NUMBER WHICH EQUALS TO ORIGINAL NUMBER

"""

for arm in range(1000,5001):

# define a logic for armstrong number

n = arm # initialization for while loop

sum\_dig = 0

while n != 0:

ind\_dig = n % 10

powers = ind\_dig \*\* 4

sum\_dig = sum\_dig + powers

n //= 10

if sum\_dig == arm:

print(arm,end = "\t")

print()

=====================================

3) WAP TO FIND THE PALINDROME NUMBERS FROM 1000 TO 3000.

4) WAP TO PRINT ALL PRIME NUMBERS FROM 100 TO 300.

**DAY19: Loop Statements Part\_03:**

INFINITE LOOPS

==============

LOOP ==> ITERATION

UNTIL THE CONDITION IS True,

THE BLOCK OF CODE ==> EXECUTE REPEATEDLY

INFINITE LOOPS ==> THE LOOP BODY, WILL EXECUTE FOR NUMBER OF TIMES

==> KEYBAORD INTERRUPT ===> CTRL + C

REASONS:

1) WHEN THE CONDITION IS INCORRECT

2) WHEN THE UPDATE IS INCORRECT

3) WHEN THE UPDATE IS MISSING

APPLICATIONS:

============

1) SERVER-CLIENT MODEL ==> PROTOCOLS (TELECOM, NETWORK)

2) GAMING APPLICATION DEVELOPMENT

======================================

TRANSFER CONTROL STATEMENTS

==============================

==> ALSO CALLED AS "LOOP CONTROL STATEMENTS"

1) WHEN WE WANT STOP EXECUTING THE BLOCK OF CODE UNCERTAINLY (UNEXPECTEDLY)

EX: 1 TO 10

1 2 3 4 5

6 ==> STOP

2) WHEN WE WANT TO PAUSE THE EXECUTION OF BLOCK OF CODE AND MAKE CONTINUE WITH THE REMAINING

EX: 1 TO 10

1 2 3 4 5

6 ==> NOT

7 8 9 10

==> TWO TYPES:

1) break statement

2) continue statement

1) break statement

===============

here: break ===> keyword

Syntax:

break

we can always define the break inside the loops only along with condition.

==> to stop executing the block of code immediately

for i in range(1,11):

if i == 6:

break

print(i,end = "\t")

======================

2) continue statement

=================

here:

continue ==> keyword

Syntax:

continue

we can always use the continue in loops only along with condition

==> when we want to pause the iteration and continue with remaining

str\_data = "Python"

for i in str\_data:

if i == 'h':

continue

print(i)

i = 100

while i >= 1:

if i % 7 == 0:

break

print(i,end = "\t")

i -= 1

i = 100

while i >= 1:

if i % 7 == 0:

i -= 1

continue

print(i,end = "\t")

i -= 1

================================

Pattern Printing

============

==> we should be aware of nested loops

lines = int(input("Enter number of rows:"))

# row wise operation

for row in range(1,lines+1):

# column wise operation

for col in range(1,row+1):

print("\* ",end = "")

print()

lines = int(input("Enter number of rows:"))

# row wise operation

for row in range(1,lines+1):

# column wise operation

for col in range(1,row+1):

print(row,end = " ")

print()

lines = int(input("Enter number of rows:"))

# row wise operation

for row in range(1,lines+1):

# column wise operation

for col in range(1,row+1):

print(col,end = " ")

print()

chr()

====

==> return a character by taking/accepting an ASCII value.

Syntax:

chr(ascii\_value)

lines = int(input("Enter number of rows:"))

value = 97 # ascii of lower case alphabet

for row in range(1,lines+1):

for col in range(1,row + 1):

alpha = chr(value)

print(alpha,end = " ")

value += 1

print()

256 characters ==> 0 to 255

digits ==> 0 to 9 ==> Ascii values ==> 48 to 56

alphabets

lower case ==> 97 to 122

upper case ==> 65 to 90

special characters

**DAY19:** **Python String Data Structure:**

for i in range(100,300):

for j in range(2,i+1):

if i % j == 0:

break

else:

print(i)

10 ==> 1, 2, 5, 10

20 ==> 1,2,4,5,10,20

================================================

DATA STRUCTURES

=================

FACEBOOK:

REGISTARTION:

NAME : STRING

USER NAME: STRING

MOBILE : INTEGER

DOB : STRING

AGE : INTEGER

GENDER : STRING

ADDRESS : STRING

MAIL : STRING

REGISTER

OUR COMPUTER ===> FACEBOOK SERVER

IN THE FORMAT OF THE TABLE

STRING DATA STRUCTURE

=======================

WHAT IS STRING

===============

A GROUP/COLLECTION OF CHARACTERS WHICH ENCLOSED WITH SINGLE QUOTES OR DOUBLE QUOTES.

EX: 'PYTHON' , "PROGRAMMING"

==> The Characters are of any type:

might be with: alphabets (any case), digits (0 to 9) and special characters etc.

STRING DEFINITION

================

==> two ways to define the strings:

1) compile time definition ==> variable can be executed with fixed data in the entire program ==> with assignment operator

2) run time definition ==> variable can be accept different values for every execution time of the program.

input(), str(input())

# string definition

# compile time definition

str\_data = 'python'

str1\_data = "Programming"

str2\_data = "123455"

str3\_data = "@#$%"

str4\_data = 'python@12343'

# run time definition of the string

str5\_data = input("Enter a string data:")

print(type(str\_data),type(str1\_data),type(str2\_data),type(str3\_data),type(str4\_data))

print(str\_data,str1\_data,str2\_data,str3\_data,str4\_data)

print(type(str5\_data))

print(str5\_data)

MULTI LINE STRING

=================

==> to define the multi-line strings:

triple quotes are used.

Ex:

'''Python is High-level,

General-purpose,

Object Oriented

Programming language.'''

"""Python is High-level,

General-purpose,

Object Oriented

Programming language."""

==> In between multi-line string,

we can allow to use:

single quotes

double quotes.

Ex:

"""Python's Features are:

'High Level'

"Object Oriented"

"""

a = ''' Python is High-level,

general purpose,

object oriented

programming language '''

b = """ Python is High-level,

general purpose,

object oriented

programming language """

c = """Python's Features are:

1) High Level Programming language.

2) 'General Purpose Programming Language.'

3) "Object Oriented Programming language."

"""

print(type(a),type(b))

print(a)

print(b)

print(type(c))

print(c)

Note:

====

1) adding of double quotes in between double quotes ==> not possible.

2) adding of single quotes in between single quotes ==> not possible

==> to add:

\start\_quotes ........ \end\_quotes

d = 'Python\'s'

e = 'Python is \'High level programming language\''

f = "python is 'High Level Programming language.'"

g = 'Python is "High level" language'

print(d)

print(e)

print(f)

print(g)

==========================================================

STRING DATA ACCESSING

======================

==> possible in two ways:

1) Indexing ==> accessing with individual characters

2) Slicing ==> accessing the part of the string with group of characters

[] ==> slice operator

1) Indexing

=========

==> with two ways:

1) using positive index

==> to access the string characters from left to right

==> called as "forward accessing"

==> default start value : 0

==> end with: length of string - 1

==> Range of positive indexing ==> 0 to length\_string - 1

==> if the index value from out of range ==> error

2) using negative index

==> to access the string characters from right to left

==> called as "reverse accessing"

==> default start value: -1

==> default end value : -length\_string

==> range of negative indexing ==> -1 to -str\_length

Syntax:

str\_data[index value]

str\_data = input("Enter a string:")

print("The Size of the string = ",len(str\_data))

# positive indexing ==> forward access

print(str\_data[0])

print(str\_data[1])

print(str\_data[2])

print(str\_data[3])

print(str\_data[4])

print(str\_data[5])

print()

# negative indexing ==> reverse access

print(str\_data[-1])

print(str\_data[-2])

print(str\_data[-3])

print(str\_data[-4])

print(str\_data[-5])

print(str\_data[-6])

2) Slicing

=======

Syntax:

str\_data[start:stop] ==> slicing start at "start" value and end with "stop-1" value

str\_data[start:stop:step] ==> slicing start with "start" value and end with "stop-1" value with the difference of "step"

str\_data[:stop] ==> by default, slicing can start with first character and end with "stop-1" value

str\_data[start:] ==> slicing can start with "start" value and end with "last" by default

str\_data[::] ==> start with first character and end with last

str\_data[::step] ==> start with first character and end with last with difference of step value.

str\_data = "Python programming"

print(str\_data[0:6])

print(str\_data[-1:-10:-1])

print(str\_data[-10:-1])

print(str\_data[0:len(str\_data)+1 : 3])

print(str\_data[0::3])

print(str\_data[::3])

print(str\_data[:-1:1])

print(str\_data[::])

print(str\_data[::-1])

print(str\_data[::4])

STRING OPERATIONS

==================

1) FINDING THE LENGTH OF THE STRING

==================================

len()

====

==> used to find the total number of characters of the given string

Syntax:

len(str\_data)

str\_data = input("Enter a string value:")

print(type(str\_data))

print("The length of the string = ",len(str\_data))

PRACTICE

**DAY20:** **String Operations Part\_02:**

STRING OPERATIONS

===================

Traversing of string data:

===================

# WAP IN PYTHON TO PRINT THE INDIVIDUAL CHARACTERS OF THE STRING ALONG WITH ITS POSITIVE AND NEGATIVE INDEX VALUE.

"""

EXPECTED OUTPUT:

THE CHARACTER AT POSITIVE INDEX IS = CHARACTER\_VALUE AND AT NEGATIVE INDEX IS = CHARACTER VALUE

"""

str\_data = input("Enter a string data:")

pindex = 0

nindex = -len(str\_data)

for i in str\_data:

# print("The Character at positive index",pindex," and at negative index",nindex,"is = ",i)

print("The Character at positive index {} and at negative index {} is = {}".format(pindex,nindex,i))

pindex += 1

nindex += 1

STRING LENGTH:

===============

len()

# WAP IN PYTHON TO FIND THE LENGTH OF THE STRING WITHOUT ANY INBUILT METHOD

str\_data = input("Enter a string data:")

str\_length = 0

for count in str\_data:

str\_length += 1

print("The Length of the given string = ",str\_length)

print("The Length of the given string = ",len(str\_data))

i = 0

str\_len = 0

while i < len(str\_data):

str\_len += 1

i += 1

print(str\_len)

==========================================

MATH OPERATIONS

=================

1) STRING CONCATENATION ==> +

=============================

==> JOINING OF TWO OR MORE COLLECTIONS (STRINGS) INTO ONE IS CALLED AS "STRING CONCATENATION".

Syntax:

str\_data1 + str\_data2 + str\_data3 + .....

str\_data1 = "Python "

str\_data2 = "is Object Oriented "

str\_data3 = "Programming Language."

print(len(str\_data1))

print(len(str\_data2))

print(len(str\_data3))

print(id(str\_data1))

print(id(str\_data2))

print(id(str\_data3))

print(str\_data1)

print(str\_data2)

print(str\_data3)

# str\_data1 = str\_data1 + str\_data2 + str\_data3

str\_data4 = str\_data1 + str\_data2 + str\_data3

print(id(str\_data1))

print(id(str\_data2))

print(id(str\_data3))

print(id(str\_data4))

print(str\_data1)

print(str\_data2)

print(str\_data3)

print(str\_data4)

===============================

2) STRING REPITITION ==> \*

========================

==> used to repeat the string data for several number of times.

'abc' \* 3 ==> 'abcabcabc'

'abcd' \* 5 ==> 'abcdabcdabcdabcdabcd'

Syntax:

str\_data \* num\_times

str\_data = input("Enter a String Data:")

print(id(str\_data))

print(str\_data)

# str\_data = str\_data \* 5

str\_data1 = str\_data \* 5

print(id(str\_data1))

print(str\_data1)

Note:

====

==> string is immutable type

for any operation, there will be the new object address.

=============================================

MEMBERSHIP CHECK ON STRING DATA:

=================================

in

not in

str\_data = "Python Programming Language"

res1 = 'Pro' in str\_data # True

res2 = 'programming' not in str\_data # True

res3 = 'pro' in str\_data

res4 = 'Programming' not in str\_data

print(res1)

print(res2)

print(res3)

print(res4)

=====================================

STRING COMPARISION

===================

POSSIBLE WITH: RELATIONAL OPERATORS

<, >, <=, >=, ==, !=

== & !=

str1 = "PyThon " # 7

str2 = "python" # 6

str3 = "PYTHON" # 6

str4 = "Python " # 7

str5 = "Pyth"

print(str1 == str2)

print(str1 == str3)

print(str1 == str4)

print(str1 != str5)

"""

Any Relational operators on the string:

1) check with length first

2) checking with individual characters of both strings wrt to ASCII value.

"""

print('pytho' > 'python')

**DAY21: String Operations Part\_03:**

STRING OPERATIONS

==================

CHANGING OF A CASE OF THE STRINGS

==================================

upper()

======

==> an inbuilt method in python,

used to convert the total given string into upper case string.

Syntax:

str\_data.upper()

lower()

=====

==> an inbuilt method in python,

used to convert the given string into lower case string.

Syntax:

str\_data.lower()

swapcase()

========

an inbuilt method in python,

used to convert a lower case string to upper case

and upper case string to lower case

Syntax:

str\_data.swapcase()

title()

====

Title case ==> that every word of the same string should be start with capital letter.

ex: python programming language

title case ==> Python Programming Language

title() is an inbuilt method in python,

used to convert any case string into title case.

Syntax:

str\_data.title()

capitalize()

========

capitalize ==> in the entire string (might be with more than one word)

that only the first word should start with capital

remaining all the characters in lower case

Ex: Python programming language

capitalize() is an inbuilt method

used to convert any case string into capitalize case.

Syntax:

str\_data.capitalize()

str1 = "python"

str2 = "Python"

str3 = "PYTHON"

str4 = "PYTHON programming LanGuage"

str5 = "object oriented programming language"

str6 = "HIGH LEVEL PROGRAMMING LANGUAGE"

# Before the case change

print(str1)

print(str2)

print(str3)

print()

# After the case change to upper case

print(str1.upper())

print(str2.upper())

print(str3.upper())

print()

# After the case change to lower case

print(str1.lower())

print(str2.lower())

print(str3.lower())

print()

# After the case change to swap case

print(str1.swapcase())

print(str2.swapcase())

print(str3.swapcase())

print(str4.swapcase())

print()

# After the case change to title case

print(str4.title())

print(str5.title())

print(str6.title())

print()

# After the case change to Capitalize case

print(str1.capitalize())

print(str2.capitalize())

print(str3.capitalize())

print(str4.capitalize())

print(str5.capitalize())

print(str6.capitalize())

================================

Note:

====

Identifiers:

variables

classes

objects

methods etc.

==> to name any entity in program, identifiers are used.

==> identifiers with different case formats

lower case, upper case, title case, capitalize, camel case

camel case:

Python programming language

camel case ==> python Programming Language

==> In Python: the identifiers can be suggested to define with "camel case" in real world.

======================================================

String Validation:

=============

Application

full name

user name

password

mail : @gmail.com/@rediffmail.com etc.

Job Portal

social media:

linkedin : url

facebook : url

endswith()

========

==> an inbuilt method in python,

used to check whether the given string data is ended with specified group of characters/string or not.

Syntax:

str\_data.endswith("reference String")

return: True/False (Boolean value)

startswith()

========

==> an inbuilt method in python,

used to check whether the given string is start with the specified group of characters/string or not.

Syntax:

str\_data.startswith("Reference\_String")

return: True/False

mail = "ravivraoinfs@gmail.com"

email = "ravivraoinfs@gmaill.com"

url = "http://www.ashokit.com"

result1 = mail.endswith("@gmail.com")

result2 = email.endswith("@gmail.com")

print(result1)

print(result2)

if url.startswith("https://"):

print("The Given URL is Secure.")

else:

print("The Given URL is not Secure.")

========================================

CHECKING OF CHARACTERS OF THE STRINGS

=======================================

isalnum()

=======

al ==> alphabets

num ==> numerals

==> an inbuilt method in python

used to check whether the given string with: alphabets and numerals only or not.

return: true/false

Syntax:

str\_data.isalnum()

================================

isalpha()

=======

==> an inbuilt method

used to check whether the given string with only alphabets or not.

return: True/False

Syntax:

str\_data.isalpha()

=======================

isdigit()

======

==> an inbuilt method

used to check whether a string with only digits or not.

Syntax:

str\_data.isdigit()

==============================

islower()

=======

==> used to check whether the given string with only lower case or not.

Syntax:

str\_data.islower()

==========================

isupper()

=======

==> used to check whether the given string with only uppercase alphabets or not.

Syntax:

str\_data.isupper()

=================================

isspace()

======

==> return True:

if the string with only spaces

False:

if the string with other than spaces

Syntax:

str\_data.isspace()

user\_name = "Ravi1234"

un1 = "ravi"

un2 = "1234"

un3 = "ravi@1234"

cn1 = "ravikumar"

cn2 = "ravi kumar"

cn3 = "ravi1234"

pin1 = "1234"

pin2 = "abcd"

pin3 = "12ab"

pwd1 = "ravi"

pwd2 = "ravi123"

pwd3 = "RAVI"

pwd4 = "r1A2v3i4"

d1 = ""

d2 = " "

d3 = " "

d4 = "ravi kumar"

d5 = " ravi "

# isalnum()

print(user\_name.isalnum())

print(un1.isalnum())

print(un2.isalnum())

print(un3.isalnum())

print()

# isalpha()

print(cn1.isalpha())

print(cn2.isalpha())

print(cn3.isalpha())

print()

#isdigit()

print(pin1.isdigit())

print(pin2.isdigit())

print(pin3.isdigit())

print()

#islower()

print(pwd1.islower())

print(pwd2.islower())

print(pwd3.islower())

print(pwd4.islower())

print()

#isupper()

print(pwd1.isupper())

print(pwd2.isupper())

print(pwd3.isupper())

print(pwd4.isupper())

print()

# isspace()

print(d1.isspace())

print(d2.isspace())

print(d3.isspace())

print(d4.isspace())

print(d5.isspace())

**DAY22:** **String Operations Part\_04:**

STRING OPERATIONS

==================

REMOVAL OF SPACE FROM THE STRING

==================================

"PYTHON PROGRAMMING" ==> SPACE NEVER REMOVED WHEN THE SPACE IN BETWEEN THE GIVEN DATA

" PYTHON"

"PYTHON "

" PYTHON "

==> REMOVAL OF SPACE FROM THE STRING IS POSSIBLE, WHEN THE STRING HAVE SPACES AT THE BEGINNING OR AT THE ENDING OR BOTH

==> THREE INBUILT METHODS:

1) strip()

2) rstrip()

3) lstrip()

1) strip()

==========

Syntax:

str\_object.strip()

==> we can remove all the spaces of the strings from beginning and from the ending

2) rstrip()

===========

==> strip operation in right side

Syntax:

str\_object.rstrip()

==> we can remove all the spaces of the string from right side only (ending)

3) lstrip()

===========

==> strip operation from left side of the string data

Syntax:

str\_obj.lstrip()

str1 = "Python Programming"

str2 = " Python"

str3 = "Python "

str4 = " Python "

print("The Given Strings are:")

print(str1)

print(str2)

print(str3)

print(str4)

str5 = str1.strip()

str6 = str2.strip()

str7 = str3.strip()

str8 = str4.strip()

str9 = str2.rstrip()

str10 = str3.rstrip()

str11 = str4.rstrip()

str12 = str2.lstrip()

str13 = str3.lstrip()

str14 = str4.lstrip()

# After the Strip operation

print("The Strings are:")

print(str5)

print(str6)

print(str7)

print(str8)

print(str9)

print(str10)

print(str11)

print(str12)

print(str13)

print(str14)

Finding of Substrings:

=====================

1) In forward direction:

========================

left to right

==> two inbuilt methods:

1) find()

2) index()

Here:

both methods can return: index value of the sub-string

1) find()

==========

Syntax:

str\_object.find('sub-string')

==> when the sub-string is at multiple places, find() can return the first occurrence

==> when the sub-string is not the part of the string, find() can return '-1'

==> using find(), we can search about the sub-string from the given range.

==> when we defined the sub-string with more than one character, find() returns: the index of First character of the sub-string only.

str\_data = "Python Programming"

# finding of single character

res1 = str\_data.find('P')

res2 = str\_data.find('b')

print(res1)

print(res2)

# finding the sub-string from the given range

res3 = str\_data.find('m',1,10)

res4 = str\_data.find('m',5,15)

print(res3)

print(res4)

# sub-string with more than one character

res5 = str\_data.find('pro')

res6 = str\_data.find('Pro')

print(res5)

print(res6)

===============================

2) index()

==========

Syntax:

str\_data.index('sub-string')

==> when the sub-string is at multiple places, index() can return the first occurrence

==> when the sub-string is not the part of the string, index() can return 'value error'

==> using find(), we can search about the sub-string from the given range.

==> when we defined the sub-string with more than one character, index() returns: the index of First character of the sub-string only.

str\_data = "Python Programming"

res1 = str\_data.index('m')

# res2 = str\_data.index("b")

res2 = str\_data.index('m',5,15)

res3 = str\_data.index("Program")

print(res1)

print(res2)

print(res3)

find() Vs index()

==================

find() can return '-1', when the sub-string is not present in the given string data.

index() can return 'value error', when the sub-string is not present in the given string data.

======================================

2) in reverse direction:

=========================

==> two inbuilt methods:

1) rfind()

2) rindex()

Syntax:

str\_data.rfind('sub-string')

str\_data.rindex('sub-string')

str\_data = "Object Oriented Programming Language"

print(str\_data.rfind('O',1,10))

print(str\_data.rfind('m'))

print(str\_data.rfind('MN'))

print(str\_data.rindex('O',1,10))

print(str\_data.rindex('m'))

print(str\_data.rindex('MN'))

========================================

# WAP IN PYTHON TO DISPLAY ALL POSITIONS OF THE SUB-STRING IN A GIVEN MAIN STRING.

data = input("Enter a string:")

subs = input("Enter a sub-string:")

flag = False

length = len(data)

pos = -1

while True:

pos = data.find(subs, pos + 1, length)

if pos == -1:

break

print("The Character at:", pos)

flag = True

if not flag:

print("The Given sub-string is not found in a main string.")

======================================================

Counting of number occurrences of the given sub-string in the main string

=========================================================

count()

=====

Syntax:

str\_data.count('sub-string')

str\_data.count('sub-string',start,stop)

str\_data = "object oriented programming language"

print("The Total occurrence of \'o\' is = ",str\_data.count('o'))

print("The Total occurrence of \'r\' is = ",str\_data.count('r'))

print("The Total occurrence of \'o\' is = ",str\_data.count('o',0,15))

=======================================

Assignment:

==========

1) WAP TO COUNT THE NUMBER OF OCCURRENCES OF ALL THE CHARACTERS OF THE GIVEN STRING.

EX:

"PYTHON PROGRAMMING"

EXP\_OUTPUT:

===========

P ==> 2

Y ==> 1

T ==> 1

**DAY23:** **String Operations Part\_05:**

STRING OPERATIONS

==================

UNICODE DEPENDENT LANGAUGE

==============================

STRING ==> WITH ALPHABETS/DIGITS/SPECIAL CHARACTERS

ALL THE CHARACTERS FOR THE STRING DEFINITION, CAN BE DEFINED WITH SOME NUMBERS ==> ASCII (AMERICAN STANDARD CODE INFORMATION INTERCAHNGE) VALUES

TOTAL OF: 256 CHARACTERS ==> 0 TO 255 ==> C/C++

UNICODE ==> 0 TO 65535

ord()

====

==> pre-defined/inbuilt method

used to get the unicode of any character

Syntax:

ord(character)

Upper Case Alphabets ==> A to Z ==> 65 to 90

Lower Case Alphabets ==> a to z ==> 97 to 122

Digits ==> 0 to 9 ==> 48 to 56

ch1 = '@'

ch2 = 'B'

ch3 = 'R'

print("The Unicode Values for the given characters are:")

u1 = ord(ch1)

u2 = ord(ch2)

u3 = ord(ch3)

print("u1 = ",u1)

print("u2 = ",u2)

print("u3 = ",u3)

chr()

==========================================

==> pre-defined method/inbuilt method

which can accept a unicode

and return a character.

Syntax:

chr(Unicode)

u1 = 65535

u2 = 0

u3 = 1234

print("The Characters according to the Unicodes are:")

ch1 = chr(u1)

ch2 = chr(u2)

ch3 = chr(u3)

print("ch1 = {} ch2 = {} ch3 = {}".format(ch1,ch2,ch3))

# WAP IN PYTHON TO PRINT ALL CHARACTERS AND ITS UNICODES

# WAP IN PYTHON TO DISPLAY ALL THE CHARACTERS BY ACCEPTING UNICODES.

print("The Characters along with Unicodes are:")

for i in range(65536):

print("The Character at {} is = {}".format(i,chr(i)))

=================================================

REPLACING OF ONE STRING WITH ANOTHER STRING

============================================

replace()

=======

Ex: "Python is Difficult programming language"

"Python is Easy programming language"

Syntax:

string\_data/string\_object.replace(old\_string,new\_string)

str1 = "Python is Difficult to learn"

print(str1)

print(id(str1))

str2 = str1.replace('Difficult','Easy')

print(str1)

print(id(str1))

print(str2)

print(id(str2))

============================

IS IT POSSIBLE TO CONVERT A STRING TO LIST

=======================================

split()

====

==> used to divide the total string into multiple pieces according to the separator

==> return an output in the list format.

Syntax:

string\_data/string\_object.split(separator\_with\_quotes)

str1 = "Python Programming Language is Easy Language"

str2 = "Python"

str3 = "07-08-2024"

str4 = "07/08/2024"

str5 = "07.08.2024"

l1 = str1.split(' ')

l2 = str1.split()

l3 = str2.split()

l4 = str3.split('-')

l5 = str4.split('/')

l6 = str5.split('.')

print(l1)

print(l2)

print(l3)

print(l4)

print(l5)

print(l6)

====================================

IS IT POSSIBLE TO CONVERT A LIST TO STRING

=======================================

join()

====

'separator'.join(list\_data/list\_object)

l1 = ['P','y','t','h','o','n']

d1 = ['07','08','2024']

s1 = ''.join(l1)

s2 = '-'.join(d1)

s3 = '/'.join(d1)

s4 = '.'.join(d1)

# print(type(s1))

print(s1)

print(s2)

print(s3)

print(s4)

==============================

# WAP TO REVERSE ORDER OF THE WORDS IN THE GIVEN STRING.

# INPUT : "PYTHON IS EASY TO LEARN"

# OUTPUT: "LEARN TO EASY IS PYTHON"

str\_data = input("Enter a string data:")

list\_data = str\_data.split()

ld = []

i = len(list\_data)-1

while i>=0:

ld.append(list\_data[i])

i -= 1

print("Reversed List = ",ld)

res\_str = ' '.join(ld)

print(res\_str)

=========================================

Assignment:

=========

1) WAP IN PYTHON TO REVERSE THE GIVEN STRING.

EX: "PYTHON"

OP: "NOHTYP"

2) WAP IN PYTHON TO REVERSE THE INTERNAL CONTENT OF EACH WORD IN A GIVEN STRING.

HINT:

IP = "ASHOK IT"

OP = "KOHSA TI"

**DAY24 : List Operations Part\_01:**

LIST DATA STRUCTURE

====================

LIST AND ITS FEATURES/PROPERTIES

================================

1) A SEQUENTIAL COLLECTION DATA ITEM

2) CAN DEFINE WITH []

AND ALL ELEMENTS IN [] MUST BE SEPARATED WITH COMMA.

EX: [ELE1,ELE2,.....]

3) PRE-DEFINED CLASS:

LIST

BECAUSE, THE LIST DATATYPE IS AN INBUILT DATATYPE.

4) List is Sequenced

because the list data can be get accessed with indexing (positive indexing/negative indexing)

5) List is supposed to accessing the part of the list using slicing

listData = [] # empty list

listData1 = [1,3,5,7,9,11]

print(type(listData))

print(type(listData1))

# positive indexing ==> forward accessing

print(listData1[0])

print(listData1[1])

print(listData1[2])

print(listData1[3])

print(listData1[4])

print(listData1[5])

print()

# negative indexing ==> reverse accessing

print(listData1[-1])

print(listData1[-2])

print(listData1[-3])

print(listData1[-4])

print(listData1[-5])

print(listData1[-6])

# slicing

print(listData1[0:6])

print(listData1[0:6:2])

print(listData1[-1:-7:-1])

print(listData1[::1])

print(listData1[::-1])

6) List can possible to define with same type of data items ==> Homogeneous List

7) List can possible to define with different type of data items ==> Heterogeneous list

8) List can ordered.

9) List can possible to define with duplicated elements.

# Homogenous List

li = [1,2,3,4,5,6,7,8,9,10] # list with integers

lf = [0.1,0.2,0.3,0.4,0.5] # list with floats

lc = [1-2j,1+2j,10-20j,10+20j] # list with complex

lb = [True, False, True, True, False] # list with booleans

ls = ['a','b','c','d'] # list with strings

print(type(li),type(lf),type(lc),type(lb),type(ls))

# Heterogeneous List

lh = [111,123,0.001,1.2e-7,True, 12-2j, 'abcde']

ld = [10,20,30,30,20,10,11,22,33,10,22,33,44,100,10,20,11,22]

print(type(lh))

print(type(ld))

print(li)

print(lf)

print(lc)

print(lb)

print(ls)

print(lh)

print(ld)

================================

HOW TO DEFINE THE LIST

======================

1) Compile Time Definition

=====================

Syntax:

Identifier = [e1,e2,e3,...]

2) Run time definition

=================

eval()

====

==> an inbuilt method

used to define any collection in the run time

Syntax:

eval(input())

lr = eval(input("Enter the list data:"))

print(type(lr))

print("The Given List = ")

print(lr)

3) Using list()

==========

Syntax:

identifier = list() ==> we can create an empty list

==> list() can create a list from other collections

list() can do type conversion

other collect ==> list collection

Syntax:

identifier = list(collection data)

4) split()

=======

strData = "Python is High Level Programming Language"

l1 = list()

l2 = list("Python")

l3 = list((1,2,3,4,5))

l4 = list({1,10,100,1000})

l5 = list({'a':10,'b':20,'c':30})

l6 = strData.split()

print(type(l1))

print(l1)

print(l2)

print(l3)

print(l4)

print(l5)

print(l6)

TRAVERSING ON THE LIST

======================

While loop:

========

ld = eval(input("Enter a list data:"))

l = len(ld)

index = 0

while index < l:

print("The Element at positive {} and at negative index {} is = {}".format(index,index-l,ld[index]))

index += 1

for loop

======

ld = eval(input("Enter a list data:"))

l = len(ld)

index = 0

for i in ld:

print("The List Element at positive index {} and at negative index {} is = {}".format(index,index-l,i))

index += 1

=========================

# WAP TO FIND THE AVERAGE OF THE LIST ELEMENTS

ld = eval(input("Enter a list:"))

s = 0

avg = 0

for i in ld:

s += i

avg = s/len(ld)

print("The Sum of List Elements = ",s)

print("The Average of List Elements = ",avg)

===================================

Membership Check:

===============

l1 = [1,3,5,7,9]

print(1 in l1)

print(100 not in l1)

List Comparision

==============

l1 = [1, 2, 3, 4, 5]

l2 = [1, 2, 3, 4, 5]

l3 = [2, 4, 6, 8]

l4 = [2, 4, 6, 8, 10]

l5 = [2,1,0,3,4]

# equality

"""

1) whether both the list datas with same length or not

2) if both list datas are with same length,

then: individual element comparison should perform

"""

print(l1 == l2) # True

print(l1 != l3) # True

print(l1 == l4) # False

# with <, >, <=, >=

"""

1) not required to check with size

2) individual element comparison should be used until the satisfactory.

"""

print(l1 > l3)

print(l1 < l4)

print(l1 <= l5)

**DAY25: List Operations Part\_02**

LIST OPERATIONS

===============

LIST IS MUTABLE:

==============

ld = eval(input("Enter a list data:"))

print(id(ld))

print(ld[0])

ld[0] = 100

print(id(ld))

print(ld)

==================================

count()

=====

==> use to find the total number of occurrences of the specified element within the list.

Syntax:

list\_data/list\_object.count(element)

==> If the specified element in count() is not the member of the list:

return = '0'

==> we cannot count the number of occurrences of the specified within the range.

ld = [1,2,3,4,5,1,3,5,7,9,1,2,3,4,5,1,3,5,7,9,2,4,6,8,10]

# if the specified element is the member of the list

print(ld.count(1))

print(ld.count(7))

# if the specified element is not the member of the list

print(ld.count(20))

# counting the element within the range, cannot count

# print(ld.count(2,1,10))

===============================================

index()

=====

==> use to find the first occurrence of the specified element in the given list.

Syntax:

list\_data/list\_object.index(element)

==> When the specified element is not a member of the list: index() returns "Value Error".

==> index() can allow to find the element within the specified range.

Syntax:

list\_data/list\_object.index(element,start,stop)

ld = [1,2,3,4,5,5,4,3,2,1,1,3,5,7,9]

# when the specified element is the member of list

# first occurrence of element is returned.

print(ld.index(9))

print(ld.index(2))

# when the specified element is not the member of list

# return: value error

# print(ld.index(100))

# finding in between range

print(ld.index(5,1,10))

=====================================

Adding of new elements into list

========================

1) append()

=========

==> use to add any element into the list at the last/end by default.

Syntax:

list\_data/list\_object.append(element)

ld = list()

print("The List before the append operation is = ")

print(ld)

ld.append(10)

ld.append(20)

ld.append(30)

ld.append(40)

ld.append(50)

print(ld)

=======================================

2) insert()

========

==> Use to add a new element at the given index position

Syntax:

list\_data/list\_object.insert(index,element)

==> if the specified index is from out of range:

the insertion of an element is at the last index automatically.

ld = [1,3,5,7,9]

print(ld)

ld.append(20)

ld.insert(1,100)

# if the specified index is exceeded the index range

ld.insert(10,1000)

print(ld)

===========================================

3) extend()

========

==> to extend a list of data to another list.

ex: l1

l2

l1 <====== l2

Syntax:

source\_list\_data.extend(destination\_list\_data)\

ld1 = [1,2,3,4]

ld2 = [5,6,7,8]

print("ld1 = ",ld1)

print("ld2 = ",ld2)

ld1.extend(ld2)

print("ld1 = ",ld1)

print("ld2 = ",ld2)

ld2.extend(ld1)

print("ld1 = ",ld1)

print("ld2 = ",ld2)

===============================================================

Note:

====

Method/Function ==> define the functionality

functionality:

take inputs

process the inputs

give an output

sum() ==> 12, 24

12 + 24

36

is count() return anything?

====================

yes,

an integer ==> count of occurrences of the specified element

is index() return anything?

====================

yes,

an integer ==> the first index value of the specified element

is append() return anything?

======================

yes,

None

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

x = ld.count(2) # 1

print(x)

y = ld.index(2)

print(y)

z = ld.append(100)

print(z)

=============================================

Removal of elements from list

=======================

1) remove()

=========

==> use to remove the specified element only from the given list.

Syntax:

list\_data/list\_object.remove(element)

==> if the specified element is unknown to the given list:

remove() return: Value Error.

ld = [1,2,3,4,5,6,7,8,9,10]

print(ld)

print(ld.remove(3))

# ld.remove(5,6)

# if the specified element is not the member of a list ==> Value Error

# ld.remove(100)

print(ld)

================================================

2) pop()

======

==> use to remove the last element from the given list automatically.

Syntax:

list\_data.pop()

==> we can use pop() to remove the specified element based on the index.

Syntax:

list\_data.pop(index)

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

print(ld)

# ld.pop()

# ld.pop()

ld.pop(1)

print(ld)

==========================================

del

===

==> del is an attribute/property

use to delete any data permanently.

Syntax:

del value\_name

==> using del property, we can delete any element from the list based on the index

==> even we can also delete the entire list also permanently.

a = 100

b = 200

l1 = [10,20,30,40,50]

print(a,b)

del a

print(l1)

del l1[2]

# print(a,b)

print(l1)

del l1

# print(l1)

**DAY26:** **List Operations Part\_03**

LIST OPERATIONS

================

DELETION OF LIST DATA:

=====================

4) clear()

=======

==> use to clear/delete the entire list

Syntax:

list\_data/list\_object.clear()

ld = [1,3,5,7,9]

print(ld)

ld.clear()

print(ld)

========================================

Reversing of list

=============

reverse()

=======

Syntax:

list\_data/list\_object.reverse()

ex: [1,2,3,4]

reverse() ==> [4,3,2,1]

# reversing of list

ld = eval(input("Enter the list data:"))

print("The List before the reverse operation is = ",ld)

ld.reverse()

print("The List after the reverse operation is = ",ld)

=========================================

Sorting of list data:

===============

Sorting ==> arranging the list elements from lower to upper value ==> Ascending order

arranging of list elements in decreasing order

sort()

====

Syntax: (for forward sorting)

list\_data/list\_object.sort()

==> can use to arrange the list items in ascending order only.

Syntax: (for reverse sorting)

list\_data/list\_object.sort(reverse = True)

# Forward sorting ===> ascending order arranging

ld = [100,1,10,7,97,79,2,6]

print("Before the sorting, the list = ")

print(ld)

ld.sort()

print("After the sorting, the list = ")

print(ld)

# reverse sorting ==> descending order arrangement

ld = [100,1,10,7,97,79,2,6]

print("Before the sorting, the list = ")

print(ld)

ld.sort(reverse = True)

print("After the sorting, the list = ")

print(ld)

=======================================

List Aliasing and Cloning

===================

when the same data with different names, all the names can pointed with same address.

ex: a = 10;b = 10;c = 10;

id(a);id(b);id(c) ==> same address

List Aliasing

=========

when the same list data, represented with two or more names

all the names(variables) pointed with the same address.

==> here, the list variables need to create from other list using 'assignment operator'

Note:

when the same list data to different variables with different definitions,

the addresses of all the variables ==> different

"When a list can define from another list using assignment operator, here: both the lists can pointed with same addresses. When any change on any list in this, it will reflect on both. This is called as "list aliasing"

List Cloning:

=========

copy()

====

==> use to create a list from another list but with the different object address.

list\_data2 = lsit\_data1.copy()

# List Aliasing

l1 = [1,3,5,7,9]

l2 = [1,3,5,7,9]

l3 = [1,3,5,7,9]

l4 = l1

l5 = l1.copy()

print(id(l1),id(l2),id(l3))

print(id(l4))

print(id(l5))

print(l1);print(l2);print(l3);print(l4);print(l5)

print()

l1[4] = 97

print(l1);print(l2);print(l3);print(l4);print(l5)

print()

l4[0] = 79

l5[1] = 97879

print(l1);print(l2);print(l3);print(l4);print(l5)

print()

================================================

Math Operations

=============

1) List Concatenation

=================

Joining of two or more lists into one ==> list concatenation

+ symbol

Syntax:

list\_data1 + list\_data2 + list\_data3

2) List Repetition

=============

To make repeat the list items for several number of times, use list repetition

symbol: \*

Syntax:

list\_data \* n.times

# Math Operations

l1 = [1,3,5,7,9]

l2 = [0,1,2,3,4]

print("Concatenated List = ",l1+l2)

print("List with repetition is = ",l1\*3)

============================

Nested List

=========

nld = [[1,2,3,4],[5,6,7,8,],[9,10,11,12],[13,14,15,16]]

print(nld[0])

print(nld[1])

print(nld[2])

print(nld[3])

print(nld[0][0])

print(nld[0][1])

print(nld[0][2])

print(nld[0][3])

for i in nld:

print(i)

print()

# Wap to take a list in list (nested list) and print the list data as matrix format.

for j in nld:

for k in j:

print(k,end = "\t")

print()

**DAY27**: **Tuple Operations Part\_01:**

TUPLE DATA STRUCTURE

======================

Tuple-Features:

============

1) It is a sequential collection datatype

2) Tuple data can define with parenthesis ()

Syntax:

identifier = (elements/items with comma separation)

3) It is a pre-defined/an inbuilt datatype

class = "tuple"

4) Tuple is an ordered datatype.

5) tuple can index supported.

to access the individual elements of the tuple, we can use an "index".

Syntax:

tuple-data-name[index-value]

Note:

====

Like strings and list:

Tuple can support both positive index and negative index.

positive indexing ==> forward access ==> accessing of tuple data from left to right

negative indexing ==> reverse access ==> accessing of tuple data from right to left.

6) Tuple can allow to access the part of the data using slicing.

Syntax:

tuple-data-name[start:stop:step]

7) Tuple can define with homogeneous elements

8) Tuple can define with heterogeneous elements

9) Tuple can be nested with other collections.

Note:

list can also be nested with with other collections.

10) Tuple can be immutable datatype.

Note:

====

to modify the tuple, we can convert that tuple into list, on this list we can define any modification.

After that the list can convert into tuple again

tuple ==> list ==> define change ==> tuple

list()

====

using this, we can convert any collection into list data.

Syntax:

list(collection)

tuple()

========

using this, we can convert any collection into tuple data.

Syntax:

tuple(collection)

a = () # empty tuple

b = (1,2,3,4) # tuple with integers

c = (1.2,0.23,1.4,0.0097) # tuple with floats

d = (True, False, True, False) # tuple with booleans

e = ('a','b','c','d') # tuple with strings

f = (111,12.234,12-24j,True,'False')

g = [(1,11,12),{13,14,15}] # list with tuple and set

h = ([1,2,3,4],{8,7,6,5}) # tuple with list and set

print(type(a));print(type(b));print(type(c));print(type(d));print(type(e));print(type(f))

print(type(g));print(type(h))

print(a);print(b)

# Positive indexing

print(b[0]);print(b[1]);print(b[2]);print(b[3]);print()

# negative indexing

print(b[-1]);print(b[-2]);print(b[-3]);print(b[-4]);print()

# slicing:

print(b[0:4:1]);print(b[::1]);print(b[-1:-5:-1]);print(b[::-1]);print()

# a[0] = 100 Type Error

lt = list(b)

lt[0] = 100

b = tuple(lt)

print(b)

TRAVERSING ON TUPLE DATA:

==========================

==> to perform the traversing on list:

two loop statements are used:

1) while loop

2) for loop

1) while loop:

================

Syntax:

initialization

while condition:

block of statements

update

# Traversing on tuple using while loop

td = (1,2,3,4,5)

index = 0

while index < len(td):

print("The Element at positive index {} and at negative index {} is = {}".format(index,index-len(td),td[index]))

index += 1

2) for loop:

=============

Syntax:

for iteration-variable in tuple-data:

block of code

td = (1,3,5,7,9)

index = 0

for p in td:

print("The Element at positive index {} and at negative index {} is = {}".format(index,index-len(td),p))

index += 1

Traversing on Tuple in reverse:

================================

# Traversing on Tuple in reverse

td = (1,2,3,4,5)

i = -1

while -len(td) <= i:

print("The Element at an index {} is = {}".format(i,td[i]))

i -= 1

===================================================================

td = (1,3,5,7,9)

index = -1

for i in td[::-1]:

print("The Element at {} is = {}".format(index,i))

index -= 1

==============================================

Creation of the tuple:

================

1) Compile time definition:

===========================

==> direct assignment of the tuple data

Syntax:

identifier = (val1,val2,val3,val4)

==> for the tuple definition, () is not mandatory.

Syntax:

identifier = val1,val2,val3,val4

Note: for the list definition [] is mandatory.

2) Dynamically changed tuple/Run time definition of tuple:

=============================================

eval()

====

we can create any collection data in run time

Syntax:

identifier = eval(input())

===================================================

3) using tuple()

================

tuple() can use to convert any collection data into tuple data

Syntax:

tuple(any collection)

# Rune Time definition of the tuple

td = eval(input("Enter a tuple data:"))

td1 = 1,2,3,4,5,6

td2 = (100,)

td3 = tuple({1,100,1000,10000})

print(type(td));print(type(td1));print(type(td2));print(type(td3))

print("The Given tuple data = ",td);print(td1);print(td2);print(td3)

==================================================

Math operations:

=============

1) Concatenation:

=================

==> joining of two or more tuples into one called as "tuple concatenation"

==> symbol: +

Syntax:

tuple1 + tuple2 + tuple3 + ....

2) Repetition:

===============

==> to make repeat the data of tuple for number of times, we can use tuple repetition.

==> Symbol: \*

Syntax:

tuple-data \* n

t1 = (1,2,3,4,5);t2 = (6,7,8,9,10);t3 = (11,13,15,17,19)

# Tuple Concatenation

t4 = t1 + t2 + t3

print("Concatenated Tuple = ",t4)

# Tuple Repetition

t5 = t1 \* 7

print("Repeated Tuple data = ",t5)

Tuple Comparison:

===============

==> it is possible with relational operators

1) with equality operators:

============================

==, !=

check:

i) length of both the tuples

=============================

if both tuple lengths are different:

equality operators return "False"

if both tuple lengths are same then: all the elements from both the tuples can be compared element by element

if all the elements are same: return "True"

otherwise: return "False"

2) checking with other relational operators:

=============================================

remaining relational operators can check and do comparison on both tuples of any length element by element

# Equality check

t1 = (1,2,3,4,5);t2 = (1,3,5,7,9);t3 = (1,2,3,4,5);t4 = (1,2,3,4,5,6);t5 = (1,3,5,7)

print(t1 == t2);print(t1 == t3);print(t1 == t4);print(t1 == t5);print()

print(t1 != t2);print(t1 != t3);print(t1 != t4);print(t1 != t5);print()

# Checking with remaining relational operators

print(t1 > t2);print(t1 < t2)

print(t1 > t4);print(t1 < t4)

**DAY28**: **Tuple Operations Part\_02:**

TUPLE OPERATIONS

==================

1) count()

========

==> an inbuilt method, which we can use to count the number of occurrences of the given of element/item in a tuple.

Syntax:

tuple-data-name.count(tuple item)

td = (1,2,3,1,2,3,4,5,6,1,2,3,4,5,6)

print(td.count(1))

print(td.count(2))

print(td.count(4))

print(td.count(100))

Note:

====

count() can return '0' if the specified element is unknown to the tuple data.

===========================================

index()

======

==> an inbuilt method which we can use to find the first occurrence of the specified element in a tuple.

Syntax:

tuple-data-name.index(tuple-element/tuple-item)

td = (1,2,3,1,2,3,4,5,6,1,2,3,4,5,6)

print("The First Occurrence of 3 in the given tuple {} is {}".format(td,td.index(3)))

# print(td.index(100)) Value error

Note:

=====

if the specified element is unknown to the tuple, index() can return: "Value Error"

sorted()

======

Note:

====

sort() ==> is not applicable to tuple

sorted() is an inbuilt method which we can use to arrange the elements in ascending order or descending order.

sorted() can arrange the tuple elements in ascending order as by default.

Syntax:

sorted(tuple-data)

Note:

====

sorted() can return a list as an output.

td = eval(input("Enter the tuple data:"))

print("The Given tuple is:")

print(td)

tds = sorted(td)

print("After the sorting, the tuple is:")

print(tuple(tds))

sorting in descending order:

=====================

Syntax:

identifier = sorted(tuple-data,reverse = True)

td = eval(input("Enter the tuple data:"))

print("The Given tuple is:")

print(td)

tds = sorted(td)

print("After the sorting, the tuple is:")

print(tuple(tds)) # ascending order

# sorting in descending order

tds1 = sorted(td,reverse=True)

print("The Tuple After the sorting is = ")

print(tuple(tds1))

===================================================

Finding of minimum and maximum of tuple data:

=============================================

min()

=====

an inbuilt method, which we can use to find the minimum of the given collection data.

Syntax:

min(collection)

max()

=====

an inbuilt method, which we can use to find the maximum of the given collection.

Syntax:

max(collection)

d1 = eval(input("Enter a string:"))

d2 = eval(input("Enter a list:"))

d3 = eval(input("Enter a tuple:"))

d4 = eval(input("Enter a set:"))

print("The Minimum value of the given collections are:")

print(min(d1));print(min(d2));print(min(d3));print(min(d4))

print("The Maximum value of the given collections are:")

print(max(d1));print(max(d2));print(max(d3));print(max(d4))

Note:

====

Python3: Tuple comparison with relational operators only possible.

Python2: cmp() can also use to do the comparison on tuples.

But, python3 cannot support cmp() for the tuple comparison.

cmp():

====

Syntax:

cmp(t1,t2)

return: 0

==> if both tuples are equal

return: 1

==> if t1 > t2

return: -1

==> if t1 < t2

=============================

Tuple Packing and Unpacking:

===========================

Packing:

========

making a tuple from individual data

Unpacking:

==========

from tuple data we can prepare individual data

# tuple packing

a = 10;b = 20;c = 30

d = 1.2;e = 1.12;f = 1.112

g = True;h = False

tp = a,b,c,d,e,f,g,h

tp1 = (a,b,c,d,e,f)

print("The Tuples are:")

print(tp);print(tp1)

# Tuple Unpacking

p,q,r,s,t,u = tp1

print("The Individual Data from tuple is = ")

print(p);print(q);print(r);print(s);print(t);print(u)

**DAY29: Set Operations Part\_01:**

SET DATA OPERATIONS

===================

COLLECTION DATATYPES IN PYTHON ARE:

1) STRING

2) LIST

3) TUPLE

4) SETS AND FROZENSETS

5) DICTIONARY

6) BYTES AND BYTE ARRAY

SET - FEATURES:

===============

1) Set is a collection datatype

2) Symbol of notation for set data is: {}

All the elements in {} must be separated with comma

ex: {1,3,5,7,9}

3) Set is an inbuilt datatype, because it has an inbuilt class: "set".

type()

4) Set is not an ordered datatype.

Ex: {1,2,3,4,5}

print() ==> {1,5,2,4,3}

5) Set is not a sequential datatype.

it is not supposed to use an index

Note: if the index is used with set ==> Type Error

6) No slicing is possible with set data.

7) set can be homogeneous

8) set can be heterogeneous.

9) No duplication is allowed on set. (no duplication can preserve by the set)

ex: {1,2,3,4,5,6,1,2,3,4,7,8,9,5,6,7}

print() ==> {1,2,3,4,56,7,8,9}

10) set can nested with only tuples

Note: limitation with list, set, dictionary

set never be nested with list

If a set with list data ==> "Type Error".

# s1 = {} # empty set

s1 = {1,3,5,7,9,11,13,15,17,19,21,23,25,100,150,200,250,300,400} # set definition homogeneous

s2 = {True, 100, 123-234j, 'string',0.001} # Heterogeneous

s3 = {1,2,3,4,5,10,20,30,40,50,1,2,3,4,5,1,3,5,7,9}

# s4 = {(1,3,5,7,9),[1,2,3,4,5],'abcde'}

# s4 = {'abcdef',(1,3,5,7,9),{1,4,6,7,9},{'a':100,'b':200,'c':300}}

s4 = {{'a':100,'b':200,'c':300},{'d':111,'e':121,'f':133}}

# s4 = {(1,2,3,4),(5,6,7,8),(9,10,11,12)}

print(type(s1));print(type(s2))

print(s1);print(s2);print(s3);print(s4)

# print(s1[1])

=========================================================

CREATION OF SETS:

================

1) COMPILE TIME DEFINITION:

===========================

Syntax:

set-object-name = {e1,e2,e3,e4,e5}

2) RUN TIME DEFINITION:

=======================

eval()

we can define the set-in run time which can dynamically change

Syntax:

eval(input("Enter a set data:"))

3) USING set() METHOD:

======================

set(): is an inbuilt method

we can use to convert/change any collection data to set data

Syntax:

set-collection-name = set() ==> to create an empty set data

set-collection-name = set(collection data)

# run time definition of set

s1 = eval(input("Enter a set data:"))

print(type(s1))

print(s1)

# using set() method

s2 = set()

s3 = set((1,3,5,7,9))

s4 = set([1,10,100,1000,11,111,1111])

print(type(s2));print(type(s3));print(type(s4))

print(s2);print(s3);print(s4)

===================================================

Traversing of set data:

=======================

Looping on set data

syntax:

for iteration-variable in set-data:

block of operation

# Traversing on sets

s1 = eval(input("Enter a set data:"))

sum = 0

for i in s1:

# print(i,end = "\t")

sum = sum + i

print("The sum of set elements = ",sum)

# index = 0

# while index < len(s1):

# print(s1[index])

# index += 1

=================================================

Math Operations:

================

1) Set Concatenation is not allowed in python.

2) Set Repetition is also not allowed.

=======================================================

Set Comparison:

===============

i) comparison with '==' and '!=' operators:

===========================================

==> these are can check lengths of the both sets

==> if both lengths are same (equal):

individual element comparison should take.

ii) comparison with '<', '>', '<=' and '>=':

=============================================

==> comparison should be based on individual elements of sets only.

s1 = {1,2,3,4,5}

s2 = {6,7,8,9,10}

s3 = {1,2,3}

s4 = {1,2,3,4,5,6}

# print(s1 + s2)

# print(s1 \* 3)

print(s1 == s2);print(s1 == s3);print(s1 == s4)

print(s1 != s2)

print(s1 > s2);print(s1 > s3);print(s1 > s4)

print(s1 < s2);print(s1 < s3);print(s1 < s4)

=====================================================

IS SET MUTABLE OR IMMUTABLE?

============================

Set is mutable

ADDING ELEMENTS INTO SET:

========================

Note: Adding elements into set using index ==> not possible

1) add()

=======

==> an inbuilt method which we can use to add an element to set.

Syntax:

existed-set = set-object-name.set(element)

Note: only one element can add to set

# using add()

s1 = {1,2,3,4,5}

print("The set before change is = ",s1)

print("The Address of set before change is = ",id(s1))

s1.add(10)

s1.add(100)

s1.add(400)

print("The set after the change is = ",s1)

print("The Address of set after the change is = ",id(s1))

2) update()

===========

==> is an inbuilt method which we can use to add multiple values to the set.

Syntax:

set-object-name.update(collection)

# using update()

s1 = {1,2,3,4,5}

print("The set before change is = ",s1)

print("The Address of set before change is = ",id(s1))

# s1.update(10) Type Error

s1.update((10,20,30))

print("The set after the change is = ",s1)

print("The Address of set after the change is = ",id(s1))

Set Aliasing:

=============

# set aliasing

s1 = {1,3,5,7,9}

s2 = s1

print(s1);print(s2)

s1.add(10)

print(id(s1));print(id(s2))

print(s1);print(s2)

3) copy()

=========

is an inbuilt method which we can use to copy the set data from other set with different object name.

Syntax:

new-set-object = old-set-object.copy()

==> set data is getting cloned.

# set aliasing

s1 = {1,3,5,7,9}

s2 = s1.copy()

print(s1);print(s2)

s1.add(100)

print(id(s1));print(id(s2))

print(s1);print(s2)

Note:

===

1) if one element is used to add in set ==> Type Error

here: update() ==> iterative method

2) if multiple individual elements using to add in set using update() ==> Type Error

Run the python file using command prompt:

========================================

cd navigation of folder

python file.py

**DAY30:** **Set Operations Part\_02:**

SET DATA OPERATIONS

===================

REMOVING OF ELEMENTS FROM SET DATA:

===================================

1) pop():

========

it is an inbuilt method which we can use to remove/delete an item/element of any from the set.

Syntax:

set-data-object.pop()

Note:

pop() can remove the element randomly.

sd1 = eval(input("Enter a set data:"))

sd2 = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15}

print("The Set data before the pop operation are = ")

print(sd1)

print(sd2)

=====================================================

2) remove():

===========

is an inbuilt method which we can use to remove/delete any specified element from the given set.

Syntax:

set-data-object.remove(element)

Note:

====

if the specified element is not in the set, then: remove() can return "Key Error".

sd1 = {1,11,21,31,41,51,61,71,81,91,101}

print("The Set before deleting is = ",sd1)

sd1.remove(21)

# sd1.remove(101,51) Type Error

# sd1.remove(97)

print("The Set after deleting is = ",sd1)

=====================================================

3) discard()

============

is an inbuilt method which we can use to remove the specified element from the given set data.

Syntax:

set-data-object.discard(element)

Note:

====

if the specified element is not known to the set, then:

discard() not return any error.

sd1 = {1,10,20,30,40,50,60,70,80,90,100}

print("The Set Before the Discard operation is = ",sd1)

sd1.discard(30)

# sd1.discard(40,50) Type Error

sd1.discard(3000)

print("The Set After the Discard operation is = ",sd1)

========================================================

4) del

=======

del is a property which we can use to delete the entire set permanently.

Syntax:

del set-data-object

Note:

====

individual element of the set cannot delete using del property.

sd1 = {1,2,10,20,3,4,30,40,5,50}

print("The Set Before the delete operations is = ",sd1)

# del sd1[0]

del sd1

# print(sd1)

====================================================

5) clear()

==========

to clear whole set by making an empty, we can use clear()

Syntax:

set-data-object.clear()

sd1 = eval(input("Enter the set data:"))

print("The Set before the clear operation is = ",sd1)

sd1.clear()

print("The Set after the clear operation is = ",sd1)

==========================================================

Set Math Operations:

====================

1) Union:

=========

Joining of two sets into one by eliminating common elements from set2.

union():

=======

Syntax:

set-object1.union(set-object2)

s1 = {1,2,3,4,5}

s2 = {4,5,6,7,8,9,10}

s3 = {6,7,8,9,10}

su1 = s1.union(s2)

su2 = s1.union(s3)

print("The Set Unions are = ")

print(su1) # {1,2,3,4,5,6,7,8,9,10}

print(su2) # {1,2,3,4,5,6,7,8,9,10}

2) Intersection:

================

can create a new set with only common elements from both the sets.

intersection()

==============

Syntax:

set-data-object1.intersection(set-data-object2)

s1 = {1,2,3,4,5}

s2 = {2,4,5,6,7,8}

si = s1.intersection(s2)

print("Set with intersection is = ",si)

==============================================================

3) Set Difference:

==================

return a set with elements from only the first set.

difference():

=============

Syntax:

set1.difference(set2)

s1 = {1,2,3,4,5}

s2 = {1,3,5,7,9,11}

print("Sets are:")

print(s1)

print(s2)

sd = s1.difference(s2)

print("All the sets are:")

print(s1)

print(s2)

print(sd)

4) Symmetric Difference:

=========================

can return a new set with elements which are in only first set and only in second set.

symmetric\_difference()

======================

Syntax:

set1.symmetric\_difference(set2)

s1 = {1,2,3,4,5}

s2 = {1,3,5,7,9}

print(s1)

print(s2)

sd = s1.symmetric\_difference(s2)

print(s1)

print(s2)

print(sd)

sd1.pop()

sd2.pop()

print("The Set data after the pop operation are = ")

print(sd1)

print(sd2)

**DAY31: Dictionary Data Operations Day\_01:**

DICTIONARY DATA OPERATIONS:

===========================

WHAT IS DICTIONARY:

==================

STRING, LIST, TUPLE, SET ==> GROUP OF ELEMENTS

'123', [1,2,3,4], (1,2,3,4), {1,2,3,4}

DICTIONARY IS A COLLECTION OF ITEMS

ITEMS ==> COMBINATION OF KEY AND VALUE PAIR

Syntax:

key : value

CAN DEFINE WITH {}

Syntax:

Identifier for dictionary object = {key1:value1,key2:value2,key3:value3}

Dictionary Properties:

======================

1) It is a collection of key and value pairs

2) It is an inbuilt datatype

predefined class: dict

type()

3) Keys in a dictionary with any datatype.

4) Values in a dictionary with any datatype.

Note: Keys in a dictionary always be with primitive datatypes.

if collection data as key in a dictionary ==> Type Error.

5) To access the dictionary values, we can use "keys".

Syntax:

dictionary-object-name[key-name]

6) If indexing can use to access the data of dictionary ==> "Key error".

If that specified index is not as a key in a dictionary.

7) Can be ordered.

8) Dictionary can be homogeneous and heterogeneous also.

homogeneous ==>

when keys with same type

and values with same type

heterogeneous

when keys ==> different type

values ==> different type

9) Mutable datatype.

10) Keys are cannot be duplicated.

11) Values can be duplicated.

# dictionary properties

d1 = {} # empty dictionary

d2 = {1,2,3,4}

d3 = {11:True, 12 : False, 13 : 112233}

d4 = {'a' : 11, 'b' : 12, 'c' : 13}

d5 = {'abc' : {1,2,3},'def' : [2,3,4],'ghi' : (9,7,8,6)}

# d6 = {[1,2,3] : 'a',(1,4,7) : 1122,{1,3,5} : 'pqrs'}

d6 = {1 : 10,2 : 20,3 :30}

d7 = {True:11,'a' : 13,1-2j : 15} # Heterogeneous

d8 = {'a' : 11,'b' : 111, 'c' : 1111} # Homogeneous

print(type(d1));print(type(d2));print(type(d3),type(d4),type(d5))

print(d1);print(d2);print(d3);print(d4);print(d5)

# values with keys

print(d3[12]);print(d4['b']);print(d5['ghi'])

print(d6[1]);print(d6[2])

# print(d6[-1])

print(d6)

print(type(d7));print(type(d8))

print(d8)

print(id(d8))

d8['c'] = 9797

print(d8)

print(id(d8))

d8['p'] = 112233

print(d8)

print(id(d8))

=============================================

Creation of Dictionary data:

============================

1) Compile time definition:

===========================

==> direct value assignment method

Syntax:

dictionary-object-name = {key1:value1,key2:value2,...}

# Compile Time Definition of the dictionary

d1 = {'a' : 100,'b' : 200, 'c' : 300,'b' : 500,'d' : 300}

print(d1)

print(type(d1))

============================================================

2) Run Time definition:

=======================

eval() ==> can use.

==> this definition can dynamically changed.

Syntax:

dictionary-object-name = eval(input("Enter a dictionary data:"))

# Run Time Definition of the Dictionary

d1 = eval(input("Enter a dictionary data:"))

print(type(d1))

print(d1)

============================================

3) using dict()

===============

Syntax:

dictionary-name = dict({key1:value1,key2:value2})

# using dict() method

d1 = dict() # empty dictionary

d2 = dict({'a' : 11,'b' : 22,'c' : 33})

# d3 = dict((1,3,5,7,9))

print(type(d1));print(type(d2))

print(d1);print(d2)

============================================================

Traversing on Dictionary:

=========================

# WAP TO ENTER NAME AND PERCENTAGE IN A DICTIONARY AND DISPLAY THAT INFORMATION.

record = dict() # created an empty dictionary

count = int(input("Enter the size of the class:"))

i = 1 # initialization

while i <= count:

name = input("Enter a student name:")

marks = input("Enter student's percentage:")

record[name] = marks

i += 1

print("Name of the Student","\t","% of marks of the Student")

for data in record:

print("\t\t",data,"\t\t\t\t\t",record[data])

==============================================================

Delete data from dictionary data:

=================================

1) using del property:

======================

# delete operation

d1 = {'abc' : 111, 'def' : 222, 'ghi' : 333, 'jkl' : 444}

# del property can use to delete the value of the dictionary with 'key

# Syntax: del dict-name[key]

print("Before The Delete Operation:",d1)

del d1['ghi']

print("After The Delete Operation:",d1)

# we can delete the whole dictionary permanently.

del d1

# print(d1)

=======================================

2) clear()

=========

it can use to delete/remove the whole dictionary.

Syntax:

dict-name.clear()

# clear() Method

d1 = {'abc' : 111, 'def' : 222, 'ghi' : 333, 'jkl' : 444}

print("The Dictionary before the delete operation:",d1)

d1.clear()

print("The Dictionary after the delete operation:",d1)

======================================

3) pop()

========

we can use to remove the value of the dictionary which is associated with the specified key.

Syntax:

dict-name.pop(key)

Note:

=====

if the specified key is not in the dictionary ==> Key Error

# pop() method

d1 = {'abc' : 111, 'def' : 222, 'ghi' : 333, 'jkl' : 444}

print("Before the pop operation:",d1)

d1.pop('ghi')

# d1.pop('pqr') Key Error

print("After the pop operation:",d1)

=====================================================

5) popitem()

============

it can use to remove an arbitrary item {key and value} from the last of the dictionary.

Syntax:

dict-name.popitem()

# popitem()

d1 = {'abc' : 111, 'def' : 222, 'ghi' : 333, 'jkl' : 444}

print("The Dictionary before the pop operation is: ",d1)

d1.popitem()

d1.popitem()

print("The Dictionary after the pop operation is: ",d1)

===========================================================

**DAY32: Dictionary Data Operations Day\_02:**

DICTIONARY OPERATIONS:

======================

1) FINDING LENGTH OF THE DICTIONARY:

===================================

len():

=====

using len() we can find the length of any collection.

Syntax:

len(dictionary-name)

# WAP TO FIND THE LENGTH OF THE DICTIONARY

d = eval(input("Enter a dictionary:"))

print("The Length of the dictionary is = ",len(d))

count = 0

for cnt in d:

count += 1

print("The Length of the dictionary is = ",count)

================================================

Getting values from the Dictionary:

===================================

dict-name[key] ==> values

get()

=====

we can use get() to access the values of the dictionary.

Syntax:

dict-name.get(key)

Note:

=====

1) If the specified key is in the dictionary ==> associated value can be returned by the get()

2) If the specified key is not in the dictionary ==> None.

3) Instead to return 'None' when the specified key is not in the dictionary

to return some default value:

Syntax:

dict-name.get(key,default-value)

d = {'apple' : 10, 'banana' : 15, 'mango' : 25, 'kiwi' : 35, 'papaya' : 20}

print(d['mango'])

res = d.get('mango')

print(res)

print(d.get('jackfruit'))

print(d.get('jackfruit',0))

=============================================

Getting of Keys from the dictionary:

====================================

keys():

======

return all the associated keys of dictionary.

Syntax:

dict-name.keys()

Getting of values from the dictionary:

======================================

values():

return all associated values of the dictionary.

Syntax:

dict-name.values()

d = eval(input("Enter a Dictionary:"))

keys = d.keys()

values = d.values()

print("All The Keys of the Dictionary = ",keys)

print("All The Values of the Dictionary = ",values)

====================================

# WAP TO PRINT ALL THE KEYS OF THE DICTIONARY

d = eval(input("Enter a dictionary:"))

for keys in d:

print(keys)

for k in d.keys():

print(k)

================================

# WAP TO PRINT ALL THE VALUES OF THE DICTIONARY

d = dict({'apple' : 100, 'banana' : 125, 'mango' : 200, 'kiwi' : 175})

for values in d:

print(d[values])

print()

for v in d.values():

print(v)

===========================================

Getting both keys and values in arbitrary from the dictionary:

==============================================================

items():

========

Syntax:

dict-name.items()

return: list of tuples with keys and values

ex: [(key1,val1),(key2,val2),(key3,val3)]

d = eval(input("Enter a dictionary:"))

items = d.items()

print(items)

for item in d.items():

print(item)

========================================

copy()

======

to create a dictionary from other dictionary, copy() can use.

Syntax:

new-dict = old-dict.copy()

# cloning of the dictionary

d = {'name' : 'Kumar', 'age' : 26, 'Course' : 'Python','Duration' : 4}

print(id(d))

print(d)

cd = d.copy()

print(id(cd))

print(cd)

cd['name'] = 'sahithi'

print(d)

print(cd)

d['name'] = 'Ashwini'

print(d)

print(cd)

===================================

Aliasing of Dictionary Data:

===========================

using assignment operator

d = {'a' : 100, 'b' : 200, 'c' : 300}

cd = d

print(d);print(cd)

print(id(d));print(id(cd))

cd['d'] = 400

d['a'] = 111

print(d);print(cd)

========================================

# WAP TO FIND THE NUMBER OF OCCURRENCES OF EACH LETTER PRESENT IN THE GIVEN STRING

# 'python'

# output: {'p' : 1, 'y' :1, 't' : 1, 'h' : 1, 'o' : 1,'n' :1}

# 'apple'

# output: {'a' : 1, 'p' : 2, 'l' : 1,'e' :1}

data = input("Enter a string data:") # apple

d = dict()

for cnt in data:

d[cnt] = d.get(cnt,0) + 1

for k,v in d.items():

print(k,"occurred",v,"times")

======================================================

Comprehensions

===============

very easy and compact way to create collection objects like: list, set, dictionary from any iterable object like: range(), list, tuple etc. based on some condition.

Syntax:

identifier = [condition for lv in collection/range] ==> List Comprehension

identifier = {condition for lv in collection/range} ==> set comprehension

identifier = {key : value for condition for lv in collection/range} ==> Dictionary Comprehension

**Note**: Tuple Comprehension is not possible in Python.

# dictionary Comprehensions

s = {k : k \* k for k in range(1,11)}

d = {i : i + 2 for i in range(10,16)}

d1 = {j : j % 2 == 0 for j in [1,2,3,4,5,6,7,8,9,10]}

print(s);print(type(s))

print(d)

print(d1)

# set comprehensions

s1 = {x \* x for x in range(6)}

print(s1)

# Tuple Comprehensions ==> not possible in python

t1 = (d \* 3 for d in range(10))

print(type(t1))

for i in t1:

print(i,end = "\t")

print()

# List Comprehension

ld = [s \* 2 for s in range(10)]

print(ld)

**DAY33:** **Functions Day\_01:**

FUNCTIONS

=========

PYTHON

FUNCTIONAL APPROACH ==> C

SCRIPTING ==> PERL

MODULAR APPROACH ==> MODULA3

calculator

arithmetic

===========

trigonometric

=============

log

====

integration etc.

===========

function can be a block of code which can be represented with some name and we can use this to perform the specific task.

==> the function cannot execute by itself.

by invoking/calling the function it can perform the task.

Advantages:

1) solving the complex or larger programs easily.

2) debugging the application is easy.

How to define the function?

===========================

keyword: def

Syntax:

def functions-name(parameters/arguments):

block of statements

How we can call the function?

=============================

Syntax:

function-name(values for parameters)

Types of Functions

==================

1) Functions without parameters and no return type

===================================================

Syntax:

def function-name():

function body

# WRITING A FUNCTION WITHOUT PARAMETERS AND NO RETURN TYPE

# function definition

def findSmall(): # function header

# function body

a = int(input("Enter a value:"))

b = int(input("Enter a value:"))

c = int(input("Enter a value:"))

d = int(input("Enter a value:"))

if a < b and a < c and a < d:

print("The Smaller value is = ",a)

elif b < c and b < d:

print("The Smaller value is = ",b)

elif c < d:

print("The Smaller value is = ",c)

else:

print("The Smaller value is = ",d)

# function call

findSmall()

2) Functions without Parameters and return type

==============================================

Syntax:

def function-name():

function-body

return value

# WRITING A FUNCTION WITHOUT PARAMETERS AND RETURN TYPE

from importlib.resources import read\_binary

def decToBin():

number = int(input("Enter a decimal:")) # 97

n = number

binary = ""

while n != 0:

rem = str(n % 2) # 1 0

binary = rem + binary # "1100001"

n //= 2

return binary

# result = decToBin()

# print(result)

print(decToBin())

==============================================

# WRITING A FUNCTION WITHOUT PARAMETERS AND RETURN TYPE

from importlib.resources import read\_binary

def decToBin():

number = int(input("Enter a decimal:")) # 97

n = number

binary = 0

power = 1

while n != 0:

rem = n % 2 # 1 0

binary = binary + rem \* power # "1100001"

power \*= 10

n //= 2

return binary

# result = decToBin()

# print(result)

print(decToBin())

3) Functions with parameters and no return type

===============================================

Syntax:

def function-name(parameter-names with comma separation):

function body

# WRITING A FUNCTION WITH PARAMETERS AND NO RETURN TYPE

def triangleArea(s1,s2,s3):

s = (s1 + s2 + s3)/2

area = (s\*(s-s1)\*(s-s2)\*(s-s3)) \*\* 0.5

print("The Area of the Triangle is = ",area)

# triangleArea(9,7,5)

# s1 = 10

# s2 = 8

# s3 = 7

#

# triangleArea(s1,s2,s3)

a = 9

b = 7

c = 8

triangleArea(c,a,b)

4) Functions with parameters and return type

============================================

Syntax:

def function-name(parameters with comma separation):

function body

return

# WRITING A FUNCTION WITH PARAMETERS AND RETURN TYPE

def typeTriangle(a,b,c):

if a == b and a == c and b == c:

return "It is an Equilateral Triangle."

elif a == b or a == c or b == c:

return "It is an Isosceles Triangle."

elif (a \* a + b \* b) == c \* c or (b \* b + c \* c) == a \* a or (a \* a + c \* c) == b\*b:

return "It is a Right Angled Triangle."

else:

return "It is a scalene Triangle."

# print(typeTriangle(10,10,10))

# a = 9

# b = 7

# c = 9

# print(typeTriangle(a,b,c))

# p = 5

# q = 4

# r = 3

# print(typeTriangle(p,q,r))

print(typeTriangle(10,11,9))

**DAY34: Functions Day\_02:**

FUNCTIONS

=========

RETURNING MULTIPLE VALUES FROM FUNCTION

========================================

# Returning Multiple Values from function

def ariths():

d1 = int(input("Enter a value:"))

d2 = int(input("Enter a value:"))

s\_add = d1 + d2

s\_sub = d1 - d2

s\_prod = d1 \* d2

return s\_add,s\_sub,s\_prod

res1,res2,res3 = ariths()

print("The Addition is = ",res1)

print("The Subtraction is = ",res2)

print("The Product is = ",res3)

=========================================

Ways to Represent the arguments in function

===========================================

4-ways to represent arguments/parameters in function:

1) positional Arguments

========================

# Positional Arguments

def function(a,b,c,d): # a,b,c and d ==> Formal Arguments

if a == b == c == d:

print("It is a Square.")

area\_square = a \*\* 2

return area\_square

elif a == c and b == d:

print("It is a Rectangle.")

area\_rectangle = a \* c

return area\_rectangle

else:

print("Not possible to calculate any")

return "None"

p = 9

q = 7

r = 9

s = 7

area = function(11,11,11,11)

area1 = function(p,r,s,q) # p,q,r and s ==> Actual Arguments

print("The Area is = ",area)

print("The Area is = ",area1)

==============================================

2) Keyword Arguments

====================

# Keyword Arguments

def printGreetings(name):

print("Hi,",name,"Good Morning")

printGreetings("Ravi")

n = input("Enter the name of the candidate:")

printGreetings(name = n) # argument is name, for the argument we have assigned value from 'n'

==============================

def rect(length,breadth):

circum = 2 \*(length + breadth)

return circum

l = int(input("Enter a length value:"))

b = int(input("Enter a breadth value:"))

result = rect(length = l,breadth = b)

res1 = rect(length = l, breadth = l)

print(result)

print(res1)

3) Default Arguments

=====================

# Default Arguments

def printGreetings(name = "user"):

print("Hi",name,",Have a great day.")

printGreetings() # without any value to the parameter

printGreetings("Rakesh")

4) Variable Length Arguments

============================

# Variable Length Arguments

def Summing(\*args):

total = 0

for i in args:

total += i

print("The Total = ",total)

Summing() # function call with 0 arguments

Summing(10) # function call with 1 argument

Summing(100,200,300,400,500)

Summing(1,2,3,4,5,6,7,8,9,10)

====================================

Scope of Variables:

===================

Two types of scopes:

1) Local Scope ==> Local Variables/function variables

=======================================================

==> are always allowed to define inside the function.

scope ==> within the same function.

# Types of Variables

def fun():

a = 10

b = 20 # local Variables

print(a,b)

=========================

fun()

# print(a);print(b)

def fun():

a = 100

b = 200

for i in range(5):

c = 10

res = i + c

print(res)

print(c)

fun()

2) Global Scope ==> Global Variables

======================================

the variables from outside the function ==> Global variables

we can access in anywhere of the program

# Global Variables

a = 10

b = 20

def fun1():

global a,b

a = 100

b = 200

print(a)

print(b)

def fun2():

print(a)

print(b)

print(a)

print(b)

fun1()

fun2()

print(a)

print(b)

=======================================

Recursion:

==========

A function that calls itself again and again is known as "recursion".

that function ===> Recursive function.

# Recursion

# Program to find the factorial of a number using recursion

"""

5! = 5 \* 4 \* 3 \* 2 \* 1

5! = 5 X fact(4)

==> 5 X 4 X fact(3)

==> 20 X 3 X fact(2)

==> 60 X 2 X fact(1)

==> 120 X 1 X fact(0)

==> 120 X 1 ==> 120

"""

def factorial(num):

if num == 0:

fact = 1

else:

fact = num \* factorial(num-1)

return fact

print("The Factorial of 5 is = ",factorial(5))

print(factorial(7))

**DAY35:** **Advanced Functions Part\_01:**

Advanced Functions:

================

1) Passing of string object as an argument/a parameter to the function:

======================================================================

Syntax:

def function-name(str-object):

function-body

# Passing of a string object to a function

def strFunction(str):

print(str)

strFunction("Hello Guys.")

strFunction("Good Morning.")

strFunction("have a great day.")

==========================================================================

# WAP IN PYTHON TO CREATE A FUNCTION WHICH ACCEPTS A STRING DATA AND CAN RETURN THE COUNT FOR THE NUMBER OF VOWELS.

def countVowels(strData):

count = 0

for ch in strData:

if ch in "AEIOUaeiou":

count = count + 1

return count

s1 = "Python Programming"

s2 = "Object Oriented Programming"

print("The Total number of vowels in {} is = {}".format(s1,countVowels(s1)))

count\_vowels = countVowels(s2)

print("The Total number of vowels in {} is = {}".format(s2,count\_vowels))

Returning of string data from function:

======================================

Syntax:

def function-name(parameter):

function-body

return string-object

# returning a string by the function

def strFunction():

a = 100

b = 200

if a < b:

smaller = a

else:

smaller = b

# result = ("The Smallest number among the given two numbers is = {}".format(smaller))

# result = 'The Smallest number among given two numbers is = ',smaller

result = "The Smallest number among the given integers is = %s"%(smaller)

return result

output = strFunction()

print(output)

======================================================

Passing of List data as an arguments to the function:

====================================================

Syntax:

def function-name(list-data-object):

function body

# passing a list data to a function

def listFunction(listData):

sum\_even = 0

for se in listData:

if se % 2 == 0:

sum\_even += se

return sum\_even

ld = eval(input("Enter some list data:"))

print("The Sum of all even elements of list {} is = {}".format(ld,listFunction(ld)))

======================================

Passing of Tuple data as an argument to the function:

====================================================

(1,2,3)

1,2,3

Syntax:

def function-name(tuple-object):

function-body

# Passing a tuple data to a function as an argument

def tupleFunction(tupleData):

for i in tupleData:

print(i,end = "\t")

td = eval(input("Enter a tuple data:"))

tupleFunction(td)

============================================

Passing of a dictionary as an argument to the function:

=========================================

Syntax:

def function-name(dictionary-data):

function-body

# Passing a dictionary as an argument to the function

def dictFunction(dictData):

print("The Name of the student is = ",dictData['name'])

print("The Roll of the student is = ",dictData['roll'])

print("The Marks of the student is = ",dictData['marks'])

student = {'name' : "Rajesh", 'roll' : 10, 'marks' : "72.90%"}

dictFunction((student))

==================================================

Assignment:

==========

1) WAP TO DEFINE A FUNCTION WHICH ACCEPTS A SET DATA AS AN ARGUMENT AND RETURN THE BIGGEST NUMBER FROM THE SET.

{1,9,7,3,5}

BIGGEST ==> 9

**DAY36:** **Advanced Functions Part\_02:**

ADVANCED FUNCTIONS:

===================

RETURNING THE LIST DATA FROM FUNCTION

=====================================

Syntax:

return list-object-name

# RETURNING A LIST FROM FUNCTION

def tupleFunction(tupleData):

listData = list(tupleData)

return listData

td = eval(input("Enter a tuple data:"))

ld = tupleFunction(td)

print("The list data i = ",ld)

====================================================

RETURNING OF DICTIONARY DATA FROM FUNCTION

==========================================

Syntax:

return dictionary-object-name

# Returning a dictionary data

def dictGenerator(name,roll,marks):

stuRecord = dict({'stuname':name,'sturoll' : roll,'stumarks' : marks})

return stuRecord

name = input("Enter the name of the student:")

roll = int(input("Enter the roll of the student:"))

marks = input("Enter the percentage marks of the student:")

record = dictGenerator(name = name,roll = roll,marks = marks)

print("The Created student record = ",record)

======================================================================

RETURNING OF TUPLE FROM FUNCTION

================================

Syntax:

return tuple-object-name

# returning a tuple from the function

def tuplePacking(a,b,c,d,e,f):

# packTuple = a,b,c,d,e,f

packTuple = (a,b,c,d,e,f)

return packTuple

td = tuplePacking(11,111,1111,11111,1111,111)

print("The Packed Tuple = ",td)

===================================================

Assignment:

===========

WAP TO CREATE A FUNCTION WHICH CAN ACCEPT THE TUPLE DATA AND PREPARE THE SET FROM THE TUPLE WITH ONLY EVEN ELEMENTS AND RETURN THAT SET DATA.

=====================================================================

SENDING A FUNCTION AS AN ARGUMENT TO THE FUNCTION

================================================

Syntax:

def function-name(function):

function-body

# PASSING A FUNCTION AS AN ARGUMENT TO THE FUNCTION

def addition(p,q):

summation = p + q

return summation

def multiplication(p,q):

product = p \* q

return product

def applyOperation(func,p,q):

return func(p,q)

result = applyOperation(addition,10,20)

print("The Summation Result = ",result)

p = applyOperation(multiplication,10,20)

print("The Multiplication result = ",p)

=================================================================

FUNCTION WITHIN FUNCTION/HIGHER ORDER FUNCTION

==============================================

Syntax:

outerFunction():

function-body

innerFunction():

function-body

# Higher Order Function

def outerFunction(text):

def innerFunction():

return text.upper()

return innerFunction()

result = outerFunction("python")

print("The String in Upper case = ",result)

Applications:

1) help organize code into more manageable pieces

2) encapsulating functionality

3) decorators

======================================================================

**DAY37:** **Advanced Functions Part\_03:**

ADVANCED FUNCTIONS

==================

FUNCTION ALIASING

=================

DEFINING THE SAME FUNCTION WITH ANOTHER NAME

def fun():

jlasksfk

fun()

name = fun

name()

fun()

# function aliasing

def printMessage(name):

print("Hey",name,"Good Morning.")

printMessage('Kumar')

# function aliasing

greets = printMessage

greets('Rakesh')

printMessage('Rahul')

wishes = greets

wishes('Ashwini')

============================================

Banking Application

xyz

abc bank

functionality-1

ANONYMOUS FUNCTION

==================

def function-name():

function-body

return

function-name()

==> anonymous function is also called as "nameless function"

keyword: lambda

Syntax:

identifier = lambda argument\_list : expression

==> lambda function can always return a value by default (without the return statement).

# anonymous function

# def square(n):

# return n \* n

#

# print(square(7))

square = lambda n : n \*\* 2

print("The Square of {} is = {}".format(9,square(9)))

why lambda function:

=====================

1) just for instant use.

2) when we want to send a function as an argument to the function, lambda function is the best option.

map(), filter(), reduce()

filter():

========

is an inbuilt function, which we can use to filter from the collection based on some condition.

syntax:

filter(function, sequence/collection)

Here:

function can use to define the condition to perform the filtering

# filter()

# WAP TO FILTER ONLY EVEN NUMBERS FROM THE LIST BY USING filter()

# def filterFunction(ld):

# rl = list()

# for i in ld:

# if i % 2 == 0:

# rl.append(i)

# return rl

#

# listData = eval(input("Enter a list:"))

#

# print("The List Even elements = ",filterFunction(listData))

==============================================================

# filter() without lambda function

def isEven(ld):

if ld % 2 == 0:

return True

else:

return False

ld = [1,10,0,101,11,100,21,20,30,31]

result = list(filter(isEven,ld))

print(result)

====================================

# filter() with lambda function

l = [0,5,10,15,20,25,30,35,40]

result = tuple(filter(lambda x : x % 2 == 0,l))

print(result)

=======================================

map() function:

===============

map() is an inbuilt method, which we can use to accept a collection/sequence

based on the condition it can modify the mapped element from the collection

Syntax:

map(function, collection)

# map()

l = [1,3,5,7,9,11]

result =list(map(lambda x : x \* x, l))

print(result)

====================================

reduce()

========

reduce() can accept a collection and reduce to the single element based on the condition.

Syntax:

reduce(function, sequence/collection)

==> to use the reduce(), we should import "functools".

Syntax:

from functools import \*

from functools import \*

l = [1,3,5,7,9,11,13,15,17,19,21,23,25]

result = reduce(lambda x,y : x + y,l)

print(result)

**Day39: Practice with Strings:**

String Practice:

================

# Practice with strings

# WAP IN PYTHON TO ACCEPT A STRING AND PRINT THE STRING REVERSE.

def reverseString(string):

# reverseString = string[::-1]

length = len(string)

index = length - 1

reverseString = ""

while index >= 0:

reverseString = reverseString + string[index]

index -= 1

return reverseString

strData = input("Enter a string:")

output = reverseString(strData)

print("The string after the reverse = ",output)

=============================================

# WAP TO TAKE A STRING AS AN INPUT AND PRINT THE STRING WITH REVERSE OF WORDS.

"""

IP = "PYTHON PROGRAMMING LANGUAGE"

OP = "LANGUAGE PROGRAMMING PYTHON

1) take an input of string

2) string ==> list, split()

3) reverse of the list data.

4) list ==> string, join()

5) Print that reverse string

"""

s = input("Enter a string data:")

ld = s.split() # converting a string into list

rev\_list = ld[::-1] # list content can be reversed

result = ' '.join(rev\_list)

print("The STring with reverse ordered words = ",result)

========================================================

# WAP IN PYTHON TO ACCEPT A STRING AND PRINT THE STRING WITH REVERSE OF THE INTERNAL CONTENT OF EACH WORD.

"""

IP : PYTHON PROGRAMMING LANGUAGE

ld = [python, programming, language]

OP : NOHTYP GNIMMARGORP EGAUGNAL

1) TAKE THE STRING ASN AN INPUT

2) STRING ==> LIST, split()

3) APPLY LOGIC FOR REVERSE OF EACH WORD/ELEMENT ON THE LIST

4) JOIN THE LIST TO STRING

5) PRINT THAT STRING

"""

s = input("Enter a string:")

ld = s.split() # string to the list

# print("The List after the split operation is = ",ld)

res\_list = []

index = 0

while index < len(ld):

res\_list.append(ld[index][::-1])

index += 1

# print(res\_list)

res\_string = ' '.join(res\_list)

print(res\_string)

======================================

# WAP TO PRINT CHARACTERS AT EVEN POSITIONS AND AT ODD POSITIONS.

s = input("Enter the string:")

print("The Characters at Even Places are:",s[0::2])

print("The Characters at Odd Places are:",s[1::2])

================================

s = input("Enter the string:")

# print("The Characters at Even Places are:",s[0::2])

# print("The Characters at Odd Places are:",s[1::2])

index = 0

s\_even = ""

s\_odd = ""

while index < len(s):

if index % 2 == 0:

s\_even = s\_even + s[index]

else:

s\_odd = s\_odd + s[index]

index += 1

print("The Characters at Even place = ",s\_even)

print("The Characters at Odd place = ",s\_odd)

=======================================

# WAP TO MERGE CHARACTERS OF 2 STRINGS INTO A SINGLE STRING BY TAKING CHARACTERS ALTERNATIVELY.

"""

S1 = "PYTHON"

S2 = "PROGRAM"

OP = PPYRTOHGORNAM"

1. define two strings with same length

2. traversing on string for merging the content

"""

s1 = input("Enter a string1:")

s2 = input("Enter a string2:")

result = ""

i,j = 0,0

while i < len(s1) or j < len(s2):

if i < len(s1):

result = result + s1[i]

i += 1

if j < len(s2):

result = result + s2[j]

j += 1

print("The String after the merge = ",result)

**Day40:** **Python Interview Questions Practice:**

Practice With Interview Questions:

==================================

# WAP TO ADD ELEMENTS TO A TUPLE.

"""

1) create the tuple

2) tuple ==> list

3) adding of elements into list

4) resultant list ==> tuple

5) print the tuple

"""

# create an empty tuple

td = tuple()

print("The Tuple before the adding elements = ",td)

# convert the tuple to list

ld = list(td)

# adding elements

for i in range(10,100,15):

ld.append(i)

print("The Resultant list = ",ld)

# resultant list ==> tuple

td = tuple(ld)

# print the tuple

print("The Tuple After adding elements is = ",td)

# WAP TO GET 4TH ELEMENT FROM THE LAST ELEMENT OF THE TUPLE.

"""

[1,2,3,4,5,6,7,8,9,10]

OUTPUT: 7

"""

td = eval(input("Enter a tuple data:"))

# print("The 4th Element of tuple is = ",td[3])

print("The 4th Element of the tuple from last element is = ",td[-4])

============================================================

# WAP TO FIND REPETED ITEMS IN A TUPLE.

# WAP TO FIND THE REPEATED ITEMS OF TUPLE.

"""

(1,2,3,4,1,2,3,4,1,1,10)

(1,2,3,4)

"""

# create the tuple

td = eval(input("Enter a tuple:"))

ld = []

for i in td:

if td.count(i) > 1:

ld.append(i)

else:

continue

for i in ld:

while ld.count(i) > 1:

ld.remove(i)

print("The Tuple with repeated elements:",tuple(ld))

========================================

# WAP TO REVERSE THE TUPLE.

# reverse of the tuple

td = (1,3,5,7,9)

rev\_tuple = tuple(reversed(td))

print(rev\_tuple)

**DAY41: OOPs Concepts Day\_01:**

OOPs:

=====

Object Oriented Programming System

Programming Language

Classified into three types:

1) Structured Programming Languages: C, C++, Python

2) Object Based : can implement anything with class and object only

ex: VB Script

3) Object Oriented: C++, Java, Python etc.

OOP Concept:

Features:

1) Class

2) Object

3) Methods

4) Constructors

5) Destructors

6) Encapsulation

7) Polymorphism

8) Inheritance

9) Overloading

10) Overriding

11) Abstraction etc.

Class:

======

Collection of Attributes and Behaviors is called as a class

Attribute ==> data in the class

Behavior ==> Method of implementation in a class

class ==> Blueprint

class definition does not require any memory

How to define the class:

========================

Keyword: class

Syntax:

class name-of-the-class:

class body

which includes: data/attributes

methods

Object:

======

==> Physical entity

use to access the data (attributes or methods) from the class.

==> an object can reserve with a memory.

Syntax:

object-name/identifier = class-name()

Accessing of an members of the class:

=====================================

Syntax:

object-name.member-name

# Class Definition

class myFirstClass:

# attributes

a = 100

b = 123.456

c = True

d = 123-234j

e = "OOPs"

# a,b,c,d,e ==> Members of the class

obj1 = myFirstClass()

print(obj1.a)

print(obj1.b)

print(obj1.c)

print(obj1.d)

print(obj1.e)

# print(a)

# Class with Collection members

class myCollectionClass:

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

at = (1,3,5,7,9)

ats = "Python"

ase = {10,20,30,40,50}

ad = {'a' : 100,'b' : 200,'c' : 300}

object = myCollectionClass()

print(object.al)

print(object.at)

print(object.ats)

print(object.ase)

print(object.ad)

======================================

Creation of More than one object for a class:

=============================================

# Class with Collection members

class myCollectionClass:

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

at = (1,3,5,7,9)

ats = "Python"

ase = {10,20,30,40,50}

ad = {'a' : 100,'b' : 200,'c' : 300}

object = myCollectionClass()

obj = myCollectionClass()

# obj1,obj2 = myCollectionClass()

obj1 = myCollectionClass()

obj2 = obj1

print(object.al)

print(object.at)

print(object.ats)

print(object.ase)

print(object.ad)

print(obj.ats)

print(obj1.al)

print(obj2.at)

=================================

Methods:

========

like functions:

we need "def" keyword to define the method.

Syntax:

def method-name(self, parameters):

method body with any

implementation

Like the functions, the methods also be invoked by using method class.

Syntax for method call:

method-name(values for the parameters)

==> Function can always define outside the class only.

==> if the function definition inside the class, we can call that definition as "method definition".

==> Method is a member of the class

Syntax to access the method of the class:

object-name.method-name(values for parameters)

==> to access/call the function, the object is not required.

==> method definition always require the self-keyword as parameter.

But the value for "self" is not required.

Syntax:

def method-name(self):

method definition

# Class with methods

def function():

print("Hello All.")

class methodClass:

def met1(self):

print("Hello, Good Morning.")

def met2(self,name):

print("Hello",name,"Good Afternoon.")

def met3(self,name):

return ("Hello {}, Good Evening.".format(name))

mo = methodClass()

mo1 = methodClass()

mo2 = methodClass()

# mo.met1('xyz')

mo.met1()

mo1.met2("Anil")

print(mo2.met3("Krishna"))

Message = mo1.met3("Ravi")

print(Message)

# mo.function()

function()

**DAY42: OOPs Concepts Day\_02:**

CONSTRUCTOR:

============

1) The constructor can define inside the class only.

2) Keyword: def

3) \_\_init\_\_()

4) The Constructor can accept the parameters/arguments

5) Constructor can be invoked automatically at the time of object creation.

6) In Python, there is only one constructor is allowed in the class.

7) Constructor can be used in python only for initialization.

8) The attributes of the class (Class variables) can always allowed to access in the constructor using "self" keyword.

Syntax:

self.var-name

# Class with constructor

class myClass:

a = 100

b = 200

# Constructor without any parameters

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

print("Hi")

print("Hello")

print("Good Morning.")

mc = myClass()

print(mc.a);print(mc.b)

# Constructor with parameters

class myClass:

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

sum\_num = a + b

print("The Sum of two numbers = ",sum\_num)

mc = myClass(100,200)

========================================

# Constructor with parameters

class myClass:

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

sum\_num = a + b

# return sum\_num

print("The sum = ",sum\_num)

mc = myClass(100,200)

====================================================

Method Vs Constructor:

=======================

1) Name of the method can be any identifier.

But the constructor name should always be "\_\_init\_\_"

2) Method need to be invoked to make the execute the method body.

But the constructor can be invoked automatically at the time of object creation.

3) Per object, we can call method for any number of times.

Whereas, per object the constructor can be called for only one time.

4) Method can allow to define with any business logic, whereas the constructor for only the initialization.

class myClass:

a = 100

b = 200

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

sum\_num = self.a + self.b

print(sum\_num)

mc = myClass()

====================================

# Constructor can be for initialization

# class myClass:

# a = 100

# b = 200

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

# sum\_num = self.a + self.b

# print(sum\_num)

#

# mc = myClass()

class myClass:

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

self.a = a

self.b = b

self.c = c

def met1(self):

print("The Values of Constructor are:")

print(self.a,self.b,self.c)

mc = myClass(10,20,30)

mc.met1()

=============================================

TYPES OF VARIABLES

==================

three types:

1) Instance Variables (Object Level Variables)

2) Static Variables (Class Level Variables)

3) Local Variables (Method Level Variables)

Instance Variables:

===================

==> Also called as "Object Level Variables".

==> The values of variables can be varied from object to object, those variables are called as "Instance Variables".

Where we can declare Instance Variables:

========================================

1) Instance Variables can declare inside the constructor using "self" keyword.

# Instance Variables

class Employee:

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

self.eid = 1221

self.ename = "Krishna"

self.edesignation = "Analyst"

self.esal = "100000"

def printData(self):

print("The Employee Id = ",self.eid)

print("The Employee Name = ",self.ename)

print("The Designation = ",self.edesignation)

print("The Salary = ",self.esal)

emp = Employee()

emp.printData()

===============================================================

2) Instance Variables can also declare inside the Instance Methods using "self" keyword.

# Instance Variables inside the Instance Method

class Test:

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

self.a = 10

self.b = 20

def met1(self): # Instance Method

self.c = self.a + self.b

def met2(self):

print("The Instance Variables are = ")

print(self.a,self.b,self.c)

t = Test()

t.met1()

t.met2()

===================================================

3) These can define in outside of the class using object reference variable.

# Instance Variables in outside the class.

class myClass:

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

self.a = 100

self.b = 200

def m1(self):

self.c = 300

mc = myClass()

mc.m1()

mc.d = 400

print(mc.\_\_dict\_\_)

==============================

year = int(input("Enter a year:"))

if year % 4 == 0 and year % 100 != 0 or year % 400 == 0:

print("This is a leap year.")

else:

print("This is not a leap year.")

**DAY43: OOPs Concepts Day\_03:**

Types of Variables:

===================

1) Instance Variables

=====================

The Instance Variables can allow to access with only object reference.

==> To Access the Instance variable, the syntax is:

object-reference.instance-variable-name

==> To delete the data/value of instance variable:

Syntax:

del object-reference.instance-variable-name

# Accessing of Instance Variables:

class Test:

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

self.a = 100

self.b = 200

def met(self):

self.c = 300

self.d = 400

t = Test()

print(t.a)

print(t.b)

t.met()

print(t.c)

print(t.d)

# deletion on instance variable

del t.a

print(t.a)

2) Static Variables

3) Method Variables

2) Static Variables

====================

the value of the variable, which is not varied from object to object is called as "static variable".

the static variables can be have only one copy for the entire class and that copy can be shared among multiple objects of the class.

static variables can be accessed with either class name or with object reference.

Syntax: (using class name)

class-name.static-variable-name

Syntax: (using object reference)

object-reference.static-variable-name

Instance Variable Vs Static variable:

======================================

Static variables can keep the same copy among different objects of the same class. Whereas, the instance variable can keep the different copies among multiple objects of the same class.

# Instance Variable Vs Static Variable

class Test:

p = 1221 # Static Variable

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

self.x = 100 # Instance Variable

t = Test()

t1 = Test()

# Before the Change:

print("t:",t.x,t.p)

print("t1:",t1.x,t1.p)

# Changing the static variable

# Test.p = 1234

Test.p = 1234

t.x = 200

# After the change

print("t:",t.x,t.p)

print("t1:",t1.x,t1.p)

==============================

Static variables can define:

1) inside the class directly but outside the from the method.

2) inside the constructor using "class name".

3) inside the method using class name.

1) inside the class directly but outside the from the method.

=============================================================

# Defining the static variable in class

class test:

x = 1221

y = 2345

t = test()

print(test.x)

print(t.y)

2) inside the constructor using "class name".

=============================================

# Defining the static variable in constructor using class name

class test:

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

test.a = 9797

test.b = 9779

t = test()

print(test.a)

print(t.b)

3) inside the method using class name.

======================================

class Test:

def met(self):

Test.p = 1221

Test.q = 9779

t = Test()

t.met()

==========================================

Local Variables:

================

also called as "method variables"

==> which are having the scope within the method.

==> method variables can always define inside the method without "self" keyword and "class name".

# Method Variables

class test:

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

self.a = 100

self.b = 200

def met1(self):

self.c = 300

test.d = 400

p = 12

q = 13

print(self.c)

print(p,q)

def met2(self):

print(self.a,self.b)

print(self.c)

print(test.d)

# print(p,q)

t = test()

t.met1()

t.met2()

t.met1()

=======================================

Types of Methods:

=================

three types:

1) Instance Methods

2) Static Methods

3) Class methods

1) Instance Methods

===================

Method with instance variables.

# Instance Methods

class Student:

def \_\_init\_\_(self,name,marks):

self.name = name

self.marks = marks

def display(self):

print("Name of the student = ",self.name)

print("Marks of the student = ",self.marks)

def grade(self):

if self.marks >= 60:

print("You Got First Grade.")

elif self.marks >= 50:

print("You Got Second Grade.")

elif self.marks >= 40:

print("You Got Third Grade.")

else:

print("You are Failed.")

stu1 = Student("Ramya",78)

stu1.display()

stu1.grade()

**DAY43: OOPs Concepts Day\_04:**

INSTANCE METHODS:

================

Setter and Getter methods:

==========================

to set instance variables

and to get instance variables ==> setter and getter methods.

Setter methods:

==============

==> use to set the instance variables

==> also called as "mutator methods"

Syntax:

def setVariable(self, variable):

self.variable = value

Getter Methods:

===============

==> use to get the instance variables

==> also called as "accessor methods"

Syntax:

def getVariable(self):

return self.variable

# Setter and Getter methods

class Student:

def setName(self,name):

self.name = name

def getName(self):

return self.name

def setMarks(self,marks):

self.marks = marks

def getMarks(self):

return self.marks

n = int(input("Enter number of students:"))

for i in range(n):

s = Student()

name = input("Enter the student name:")

s.setName(name)

marks = int(input("Enter marks of the student:"))

s.setMarks(marks)

print("Hi,",s.getName())

print("Your Marks are:",s.getMarks())

====================================================

Class Methods:

==============

==> the methods with only class variables (static variables) called as "class methods".

@classmethod

==> class methods are always can call with "class name" or "object reference".

# Class Methods

class Animal:

legs = 4 # static variable

def m1(self):

print("This is the Animal class.")

@classmethod

def walk(cls,name):

print("{} walks with {} legs".format(name,cls.legs))

Animal.walk("Dog")

am = Animal()

am.walk("Cow")

=========================================================

Static Methods:

===============

==> are called as "general utility methods"

==> inside the static methods, we won't include, class variables and instance variables.\

the variables with "self" and with "cls" were not in static method.

==> static method, not required "self" or "cls" as parameter in the definition.

==> annotation:

@staticmethod

==> calling of static method:

use: class name or object reference

# static methods

class Calculator:

@staticmethod

def add(x,y):

print("The Sum = ",x+y)

@staticmethod

def multy(x,y):

print("The Product = ",x\*y)

Calculator.add(100,200)

cl = Calculator()

cl.multy(23,27)

======================================

Passing of members of one class to another class:

=================================================

cls1:

mem1

mem2

cls2:

mem2

# Passing the members of one class to another class.

class Employee:

def \_\_init\_\_(self,eno,ename,esal):

self.eno = eno

self.ename = ename

self.esal = esal

def display(self):

print("The Id of the employee = ",self.eno)

print("Name of the employee = ",self.ename)

print("Salary of the Employee = ",self.esal)

class Test:

def modify(emp):

emp.esal = emp.esal + 100000

emp.display()

e = Employee(101,"Radhika",120000)

e.display()

Test.modify(e)

====================================

Inner Classes:

==============

# Inner class

class Outer:

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

print("Outer class Object Creation.")

class Inner:

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

print("Inner class Object creation.")

def m1(self):

print("This is inner class method.")

o = Outer()

i = o.Inner()

i.m1()

**DAY44: OOPs Concepts Day\_05:**

Encapsulation:

==============

what is Encapsulation?

=======================

1) Core concept of OOP.

2) Basic Idea of encapsulation is:

The wrapping up/Bundling of data members and methods of the class into the single unit.

3) Encapsulation in Python also ensures that objects are self-sufficient functioning pieces and can work independently.

Why do we need Encapsulation?

=============================

1) Provide well-defined and reusable code

2) Prevents the accidental modifications or deletion.

3) Provides the security

4) reusability

Access modifiers:

================

are use to restrict or provide the limited access to certain variables/data and/or methods while programming.

==> three types of access modifiers:

1) Public Members

2) Private members

3) Protected Members

Note:

Like java, there are no keyword (public, private and protected) in python.

Encapsulation with Public members:

==================================

Public members can be accessed anywhere within the class or from any part of the program.

Note:

all the class members are "public" type as default.

# Encapsulation with Public access members

class publicModifiers:

def \_\_init\_\_(self,name,age):

# name and age ==> local to the constructor

# convert the local members to instance variables

self.name = name

self.age = age

def Age(self):

# accessing public modifier data

print("Age of the person is = ",self.age)

def call(self):

self.Age()

pmo = publicModifiers("Ravi",31)

# accessing of public access data from outside the class but within the program

print("The Name of the person is = ",pmo.name)

pmo.Age()

pmo.call()

================================================

Encapsulation with Private members:

===================================

private members ==> \_\_

the name of the data should be prefixed with '\_\_' ==> are called as "private members".

ex: \_\_a, \_\_p etc.

# Encapsulation with Private members

class Rectangle:

# private members

\_\_length = 0

\_\_breadth = 0

\_\_area = 0

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

self.\_\_length = 7

self.\_\_breadth = 9

# Printing of private data in the constructor

print("The Length = ",self.\_\_length)

print("The Breadth = ",self.\_\_breadth)

def Area(self):

self.\_\_area = self.\_\_length \* self.\_\_breadth

print("The Area = ",self.\_\_area)

rect = Rectangle()

# accessing private data from outside the class

# print("The Length = ",rect.\_\_length)

# print("The Breadth = ",rect.\_\_breadth)

rect.Area()

==============================================

Encapsulation with Protected Members:

=====================================

protected members ==> \_

ex: \_a, \_b etc.

# ENCAPSULATIOON WITH PROTECTED MEMBERS

class details:

# protected members

\_name = "Kumar"

\_age = 28

\_job = "Developer"

class modification(details):

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

print("Name = ",self.\_name)

print("Age = ",self.\_age)

print("Designation = ",self.\_job)

obj = modification()

obj1 = details

print("Name = ",obj1.name)

print("Age = ",obj1.age)

print("Designation = ",obj1.job)

**DAY45: OOPs Concepts Day\_06:**

Why Static Methods?

===================

@staticmethod

def stattcimethodname(self,par1,par2,..):

definition

X-app:

class app1:

met1:

definition

met2:

definition

class app2:

met3:

definition

met4:

definition

met1:

static methods allow you to isolate the utility methods into different sub-modules.

Why class method?

================

@classmethod

def classmethodname(cls,par1,par2):

implementation

X-app:

class a1:

@classmethod

met1:

implementation

met2:

definition

class a2:

class method can bound to the class but not to the instance of the class.

======================================

Garbage Collection:

===================

C++ ==> OOP Concepts

class

object

create : for need

destruct : which are not in need

memory issues for the application to run

Python, is an assistant which is called "garbage collector"

can use to destruct the objects automatically when are not in use.

Enabling and Disabling Garbage Collector:

=========================================

In python, the garbage collector can be enabled automatically.

To disable the garbage collector:

1) gc.isenabled()

used to check whether the garbage collector is enable or not.

==> True ==> if the garbage collector is in enable state.

==> False ==> if the garbage collector is in disable state.

2) gc.disable() ==> for disabling of garbage collector.

==> True ==> if the garbage collector is disabled

==> False ==> otherwise

3) gc.enable()

==> we should import garbage collector.

Syntax:

import gc

import gc

# checking of garbage collector, whether it is in enable state or not

print(gc.isenabled())

# disabling of garbage collector

gc.disable()

print(gc.isenabled())

# enabling of garbage collector

gc.enable()

print(gc.isenabled())

Ex:

class abc:

m1:

m2:

o1 = abc() ==> 1kb

o2 = abc() ==> 1kb

o3 = abc() ==> 1kb

o1.m1()

o1.m2()

have a memory of 10kb ==> abc

o1.m1() ==> 2kb

o1.m2() ==> 2kb

remaining ==> 10-7 ==> 3kb

02.m2() ==> 2kb

01.m1() ==> 2kb ==> error ==> out of memory

garbage collector

03 ==> free ==> 1kb

smart phone ==> playstore

gaming app ==> 80mb ==> install, on our phone, 80 mb of memory is reserved for the app

f1 ==> obj1

f2 ==> obj2

f3 ==> obj3

f4 ==> obj4 ==> 20mb

f5 ==> obj5

f5()

f5()

80

**DAY46: OOPs Concepts Day\_07-PartI:**

POLYMORPHISM:

=============

POLYMORPHISM ==> POLY MORPHS

POLY ==> MANY

MORPHS ==> FORMS

POLYMORPHISM ==> MANY FORMS

EX:1) PERSON'S BEHAVIOR

PERSON CAN BEHAVE AT THEIR PARENT ==> ONE WAY

SAME PERSON CAN BEHAVE AT THEIR FRIENDS ==> SECOND WAYS

ONE PERSON WITH MULTIPLE BEHAVIORS ==> POLYMORPHISM

EX:2)

+ ==> SUM/ADDITION

120 + 240 = 360

+ ==> CONCATENATION

'A' + 'B' ==> 'AB'

OPERATOR ==-> CAN PERFORM TWO OPERATIONS ==> POLYMORPHISM

3) MET1():

RETURN A+B

MET1():

RETURN A+B+C+D

POLYMORPHISM CAN DEFINE WITH:

1) DUCK TYPING PHILOSOPHY OF PYTHON

2) OVERLOADING

OPERATOR OVERLOADING

METHOD OVERLOADING

CONSTRUCTOR OVERLOADING

3) OVERRIDING

METHOD OVERRIDING

CONSTRUCTOR OVERRIDING

DUCK TYPING PHILOSOPHY OF PYTHON

================================

c, JAVA

=======

int var = 10;

void f1(int obj)

{

obj

}

python:

========

var = 100

def f1(obj):

obj.talk()

f1(12) ==> obj ==> int

class Duck:

def talk(self):

print("Quack... Quack")

class Dog:

def talk(self):

print("Bow... Bow...")

def f1(obj):

obj.talk()

# same object for multiple classes

obj = [Duck(), Dog()]

for i in obj:

f1(i)

=============================================

Overloading

===========

+ ==> sum

+ ==> concatenation

m1():

a+b

m1():

a\*b

deposit()

deposit(cash)

deposit(cheque)

deposit(dd)

one thing with multiple definitions/implementations ===> overloading

==> three types:

1) Operator Overloading

2) method overloading

3) Constructor overloading

Operator Overloading

====================

class simpleInterest:

def \_\_init\_\_(self,principle,rate,time):

self.principle = principle

self.rate = rate

self.time = time

def met1(self):

self.si = (self.principle \* self.rate \* self.time)/100

total = self.si + self.principle # adding

result = "The Total Amount is = " + str(total) # concatenation

return result

sobj = simpleInterest(10000,3,3)

print(sobj.met1())

========================================

class Book:

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

self.pages = pages

def \_\_add\_\_(self, other):

return self.pages + other.pages

b1 = Book(100)

b2 = Book(200)

print(b1 + b2)

===============================

class Book:

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

self.pages = pages

def \_\_add\_\_(self, other):

return self.pages + other.pages

def \_\_mul\_\_(self, other):

return self.pages \* other.pages

def \_\_sub\_\_(self, other):

if self.pages > other.pages:

return self.pages - other.pages

else:

return other.pages - self.pages

b1 = Book(100)

b2 = Book(200)

print(b1 + b2)

print(b1 \* b2)

print(b1 - b2)

===============================

WAP IN PYTHON TO OVERLOAD > AND <=.

> ==> \_\_gt\_\_(self,other):

<= ==> \_\_le\_\_(self,other)

===================================

Employee salary:

HRA

DA

TA

Gratuity

Paid leaves

salary

my salary is + "121200.00"

**DAY47: OOPs Concepts Day\_07-PartII:**

POLYMORPHISM:

=============

Poly ==> Many

Morphs ==> Forms

Polymorphism ==> Many forms

one thing in many forms is called polymorphism.

ex:

digital calculator

addition:

two numbers ==> 1 + 2 ==> 3

three numbers ==> 10 + 20 + 30 ==> 60

four numbers

+ ==> addition

+ ==> concatenation

\* ==> multiplication

\* ==> repetition

==> Polymorphism can perform in different ways:

1) Function with class object

2) Overloading

3) Overriding

1) Function with class object

==============================

# Polymorphism

# method-1: Function with class object

class Class1:

def m1(self):

print("Hello")

print("Good Morning.")

class Class2:

def m1(self):

print("Hi")

print("Have a great day.")

def fun(obj):

obj.m1();

# obj1 = Class1()

# obj1.m1()

#

# obj2 = Class2()

# obj2.m1()

# to create a common object for multiple classes

obj = [Class1(),Class2()]

# for o1 in obj:

# fun(o1)

# obj.m1()

obj[0].m1()

====================================

OVERLOADING

===========

def met1(self):

a+b

def met1(self):

a+b+c

def met1(self):

a+b+c+d

==> three types:

1) Operator Overloading

2) Method Overloading

3) Constructor Overloading

Operator Overloading:

=====================

+ ==> sum

+ ==> string concatenation

# Operator Overloading

class Student:

def \_\_init\_\_(self,m1,m2,m3):

self.m1 = m1

self.m2 = m2

self.m3 = m3

def calci(self):

self.total = self.m1 + self.m2 + self.m3 # addition

result = "The Total of Student = "+str(self.total) # concatenation

print(result)

stu = Student(88,79,97)

stu.calci()

================================================

# Operator Overloading

class Calculator:

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

self.a = a

self.b = b

def m1(self):

self.res = self.a + self.b

return self.res

def \_\_add\_\_(self,other):

return self.res + other.res

def \_\_mul\_\_(self, other):

return self.res \* other.res

obj1 = Calculator(10,20)

obj2 = Calculator(20,30)

print(obj1.m1())

print(obj2.m1())

print(obj1 + obj2)

print(obj1 \* obj2)

==================================

Assignment:

=========

WAP in python to overload the less than and greater than operators.

=======================================================

Method Overloading:

===================

# Method Overloading

# class methodOverload:

# def m1(self):

# print("With zero parameters.")

#

# def m1(self,a):

# print("With one parameter")

#

# def m1(self,a,b):

# print("With two parameters")

#

# mo = methodOverload()

#

# # mo.m1() # no arguments

# # mo.m1(10)

# mo.m1(10,20)

class methodOverloading:

def addition(self,a = None,b = None,c = None):

if a != None and b != None and c != None:

self.res = a + b+ c

print("The Sum of three numbers = ",self.res)

elif a != None and b != None:

self.res = a + b

print("The Sum of two numbers = ",self.res)

elif a != None:

self.res = a

print("The Sum of one number = ",self.res)

else:

self.res = 0

print("The Sum is = ",self.res)

mo = methodOverloading()

mo.addition() # with 0 arguments

mo.addition(10) # with 1 argument

mo.addition(10,20) # with two arguments

mo.addition(10,20,30) # with three arguments

=====================================================

Constructor Overloading

========================

Constructor overloading is not possible in Python.

# Constructor Overloading

class Class1:

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

print("The Constructor with 0 parameters.")

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

print("The Constructor with 1 parameter.")

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

print("The Constructor with 2 parameters.")

# obj = Class1()

# obj = Class1(10)

obj = Class1(10,20)

**DAY48: OOPs Concepts Day\_08:**

POLYMORPHISM:

=============

Method Overloading:

===================

m1():

implement-1

m1():

implment-2

m1():

implment-3

m1():

without arguments

m1(self,a):

with one parameter

m1(self,a,b):

with two parameters

len():

len("Python") ==> 6 ==> total number of characters

len(["Python","Abcde","Defghi""JKLMN"]) ==> 4 ==> count of number of string elements

len({"abc":"pqr","def":"xyz","ghi":"lmn"}) ==> 3 ==> count of number of pair of strings

ex:

digital calculator:

sum():

NO DATA

sum = 0

sum(10)

SUM = 10

SUM(10,20)

SUM = 30

BNKING APPLICATION

ROI

SIMPLE

COMPUND

class Calculator:

def addition(self,a = None,b = None,c = None,d = None,e = None):

if a != None and b != None and c != None and d != None and e != None:

print("The sum of 5 numbers = ",a+b+c+d+e)

elif a!= None and b != None and c != None and d != None:

print("The Sum of 4 numbers = ",a+b+c+d)

elif a != None and b != None and c != None:

print("The Sum 3 numbers = ",a+b+c)

elif a != None and b != None:

print("The Sum of 2 numbers = ",int(a)+int(b))

elif a != None:

print("The Sum is = ",a)

else:

print("The Sum = ",0)

cls = Calculator()

cls.addition() # 0 parameters

cls.addition(100) # 1 parameter

cls.addition(121,122)

cls.addition(111,222,333)

cls.addition(12,24,36,48)

========================================

# method overloading

class greetings:

# def wish(self):

# print("Good Morning")

# def wish(self):

# print("Good Afternoon")

# def wish(self):

# print("Good Evening")

def wish(self,time = 0):

if time > 0 and time < 12:

print("Good Morning")

elif time >= 12 and time < 16:

print("Good Afternoon")

elif time >= 16 and time < 20:

print("Good Evening.")

elif time >= 20 and time < 24:

print("Good Night.")

else:

print("time Zone is searching..")

# when the same method with three definitions,

# the last definition is considered as the updated definition to execute.

gt = greetings()

gt.wish() # 0 parameters

gt.wish(6)

gt.wish(13)

gt.wish(22)

Constructor Overloading:

=======================

No constructor overloading is supported by the python.

# constructor overloading

class constructorOverloading:

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

print("Hello")

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

print("Hello",name)

# obj = constructorOverloading()

obj = constructorOverloading("Kumar")

**DAY49: OOPs Concepts Day\_08:**

POLYMORPHISM:

=============

Poly ==> Many

Morphs ==> Forms

Polymorphism ==> Many forms

one thing in many forms is called polymorphism.

ex:

digital calculator

addition:

two numbers ==> 1 + 2 ==> 3

three numbers ==> 10 + 20 + 30 ==> 60

four numbers

+ ==> addition

+ ==> concatenation

\* ==> multiplication

\* ==> repetition

==> Polymorphism can perform in different ways:

1) Function with class object

2) Overloading

3) Overriding

1) Function with class object

==============================

# Polymorphism

# method-1: Function with class object

class Class1:

def m1(self):

print("Hello")

print("Good Morning.")

class Class2:

def m1(self):

print("Hi")

print("Have a great day.")

def fun(obj):

obj.m1();

# obj1 = Class1()

# obj1.m1()

#

# obj2 = Class2()

# obj2.m1()

# to create a common object for multiple classes

obj = [Class1(),Class2()]

# for o1 in obj:

# fun(o1)

# obj.m1()

obj[0].m1()

====================================

OVERLOADING

===========

def met1(self):

a+b

def met1(self):

a+b+c

def met1(self):

a+b+c+d

==> three types:

1) Operator Overloading

2) Method Overloading

3) Constructor Overloading

Operator Overloading:

=====================

+ ==> sum

+ ==> string concatenation

# Operator Overloading

class Student:

def \_\_init\_\_(self,m1,m2,m3):

self.m1 = m1

self.m2 = m2

self.m3 = m3

def calci(self):

self.total = self.m1 + self.m2 + self.m3 # addition

result = "The Total of Student = "+str(self.total) # concatenation

print(result)

stu = Student(88,79,97)

stu.calci()

================================================

# Operator Overloading

class Calculator:

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

self.a = a

self.b = b

def m1(self):

self.res = self.a + self.b

return self.res

def \_\_add\_\_(self,other):

return self.res + other.res

def \_\_mul\_\_(self, other):

return self.res \* other.res

obj1 = Calculator(10,20)

obj2 = Calculator(20,30)

print(obj1.m1())

print(obj2.m1())

print(obj1 + obj2)

print(obj1 \* obj2)

==================================

Assignment:

=========

WAP in python to overload the less than and greater than operators.

=======================================================

Method Overloading:

===================

# Method Overloading

# class methodOverload:

# def m1(self):

# print("With zero parameters.")

#

# def m1(self,a):

# print("With one parameter")

#

# def m1(self,a,b):

# print("With two parameters")

#

# mo = methodOverload()

#

# # mo.m1() # no arguments

# # mo.m1(10)

# mo.m1(10,20)

class methodOverloading:

def addition(self,a = None,b = None,c = None):

if a != None and b != None and c != None:

self.res = a + b+ c

print("The Sum of three numbers = ",self.res)

elif a != None and b != None:

self.res = a + b

print("The Sum of two numbers = ",self.res)

elif a != None:

self.res = a

print("The Sum of one number = ",self.res)

else:

self.res = 0

print("The Sum is = ",self.res)

mo = methodOverloading()

mo.addition() # with 0 arguments

mo.addition(10) # with 1 argument

mo.addition(10,20) # with two arguments

mo.addition(10,20,30) # with three arguments

=====================================================

Constructor Overloading

========================

Constructor overloading is not possible in Python.

# Constructor Overloading

class Class1:

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

print("The Constructor with 0 parameters.")

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

print("The Constructor with 1 parameter.")

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

print("The Constructor with 2 parameters.")

# obj = Class1()

# obj = Class1(10)

obj = Class1(10,20)

**DAY 50: OOPs Concepts Day\_08 -Part2:Inheritance:**

Inheritance:

============

phonepe{

send money

{

mobile

upi

account

{

}

check balance

{

implementation

}

}

ex:

p1 =====> p2

phonepe

mobile

{

mobile

amount : 10000

upi

proceed

check balance

}

Inheritance:

============

Acquiring of data and methods from one class to another class is called as "Inheritance".

Use:

====

reusability of code can be increased.

4-types:

1) Single Inheritance ==> 1-1 inheritance

class Parent:

def m1(self):

print("Hello Parent.")

class Child(Parent):

def m2(self):

print("Hello Child.")

po = Parent()

co = Child()

po.m1()

co.m2()

co.m1()

# po.m2()

==> Here:

the object of child class, can inherit the data from both parent class and child class also.

Whereas, the object of parent class can inherit the data from only parent class (cannot from child class).

2) Multiple Inheritance ==> many to one

==> In this,

there are two independent parent classes (not inherit to each other)

and one child for both parent classes.

==> in this case,

the object of child can inherit the data from parent-1, parent-2 and from itself.

Whereas, the object of parent-1 can access the data of itself and same to the parent2 also.

class Parent1:

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

self.a = a

self.b = b

def addition(self):

self.summation = self.a + self.b

print("The Sum of Two numbers = ",self.summation)

class Parent2:

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

self.a = a

self.b = b

def subtraction(self):

self.difference = self.a - self.b

print("The Difference of two numbers = ",self.difference)

class Child(Parent1,Parent2):

def printing(self):

print("The Results of Sum and Difference are:")

# parent class objects

# po1 = Parent1(100,200)

# po2 = Parent2(200,300)

# object of child class

co = Child(1000,2000)

co.printing()

co.addition()

co.subtraction()

# parent classes are independent together

po1 = Parent1(200,100)

po1.addition()

# po1.printing()

# po1.subtraction()

3) Multilevel Inheritance

==> Grand Parent class ==> Parent Class ==> Child Class

the object of child class, can inherit the data from: itself, parent and from Grand Parent class also.

the object of parent class, can inherit the data from itself and from grand Parent class.

the object of grand parent class, can inherit from its own.

class Parent1:

def f1(self):

print("Good Morning.")

class Parent2(Parent1):

def f2(self):

print("Good Afternoon.")

class Child(Parent2):

def f3(self):

print("Good Evening.")

co = Child()

co.f3()

co.f2()

co.f1()

p1o = Parent2()

p1o.f2()

p1o.f1()

4) Hierarchy Inheritance

==================================================

Overriding:

===========

m1():

adding

m1():

product

m1():

sub

1) Method Overriding

# class A:

# def f1(self):

# print("The Method f1 in Class 'A'")

#

# def f2(self):

# print("The Method f2 in Class 'B'")

#

# class B(A):

# def f1(self):

# print("I am Redefining the method f1 in Child Class B")

#

# def f2(self):

# print("I am Redefining the method f2 in Child Class B")

#

# # child = B()

# # child.f1()

# #

# # A().f1()

#

# if(\_\_name\_\_ == "\_\_main\_\_"):

# child = B()

#

# child.f2()

#

class parent:

def f1(self):

print("Hi")

def f1(self):

print("Hello")

def f1(self):

print("Have a Great day.")

po = parent()

po.f1()

**DAY51: OOPs Concepts Day\_09:**

DATA ABSTRACTION:

=================

x-app:

functionality-1:

data

methods ==> hide

implementation ==>

functionality-2:

data

methods

implementation

functionality-3:

data

methods

implementation

phonepe

check balance

implementation

money transfer

mobile

data

method

implementation

upi

account

TV

==> Remote controller

channel up/down

volume up/down

turnoff

turn on

==> The Data or methods in a class, we can give permission to access if required. We can hide if not necessary. This operation is called as "Data Abstraction".

Why?

===

1) For security

2) Reusability is increased.

Class is instantiated.

class ClassName:

data

method

obj = ClassName()

obj.data

obj.method

Abstract class:

the class cannot be instantiated.

but meant to be inherited.

from abc import ABC, abstractmethod

class abstractClass(ABC):

@abstractmethod

def myMethod(self):

pass

"""

to add an abstract method inside the abstract class,

there is an annotation/decorator "@abstractmethod"

"""

obj = abstractClass()

obj.myMethod()

from abc import ABC, abstractmethod

class Parent(ABC):

# instance method

def met1(self):

print("This is the common method of the class 'Parent'")

@abstractmethod

def met2(self):

print("Hello")

class Child1(Parent):

def met2(self):

print("The Abstract method of Parent Class is defined in Child1 class.")

class Child2(Parent):

def met2(self):

print("The Abstract method of Parent Class is defined in Child2 class.")

# obj1 = Parent()

#

# obj1.met1()

obj1 = Child1()

obj1.met2()

obj2 = Child2()

obj2.met2()

================================

from abc import ABC, abstractmethod

class Greetings(ABC):

def general(self,name):

print("Hi",name)

@abstractmethod

def wishes(self):

print("Welcome")

class timeZone1(Greetings):

def wishes(self):

print("Good Morning.")

class timeZone2(Greetings):

def wishes(self):

print("Good Afternoon.")

class timeZone3(Greetings):

def wishes(self):

print("Good Evening.")

# abstract classes are not instantiated

# tz1 = timeZone1()

#

# tz1.general("Ravi")

# tz1.wishes()

# tz2 = timeZone2()

# tz2.general("Kumar")

# tz2.wishes()

tz3 = timeZone3()

tz3.general("Ashwini")

tz3.wishes()

=====================================================================================

**DAY52: Modules:**

group of functions, variables and classes into a file ==> a module.

==> .py file ==> a module

{

functions,

variables,

classes

objects

}

Module:

=======

x = 1234 # variable

def add(p,q): # function-1

s = p + q

return s

def product(x,y): # function-2

m = x \* y

return m

class employee: # class

def \_\_init\_\_(self,name,role,age,salary):

self.name = name

self.role = role

self.age = age

self.salary = salary

def employeeData(self):

print("The Details of the Employee:")

print("Name = ",self.name)

print("Disgnation = ",self.role)

print("Age = ",self.age)

print("Salary = ",self.salary)

==========================

practice file:

==============

import mod11

# Accessing the variable of mod11

# syntax: module-name.variable-name

print(mod11.x)

# checking of even or odd

if mod11.x % 2 == 0:

print(mod11.x,"is an even number.")

else:

print(mod11.x,"is an odd number.")

# accessing of the functions

# syntax: module-name.function-name(val1, val2, val3,..)

result1 = mod11.add(1234,3456)

result2 = mod11.product(26,27)

print("The Sum = ",result1)

print("The Product = ",result2)

# Accessing of the class

# creating the object

obj1 = mod11.employee("Sahithi","Software Engineer", 22, 600000)

obj1.employeeData()

============================================

Importing the module with different name:

=========================================

import mod11 as calculation

# accessing variable

print(calculation.x)

# access the functions

print(calculation.product(10,20))

# create the object for the class of mod11 using alias name

obj = calculation.employee("Kumar", "Sr. SOftware Engineer", 28, 1500000)

obj.employeeData()

Importing of Selected member from the module

============================================

Syntax:

from module-name import member-name, member-name1,...

to access the members:

Syntax:

member-name

# importing the selected member from the class

from mod11 import employee,add

# syntax: member-name

# object-name = class-name()

emp = employee("Rahul","Software Developer", 29, 1600000)

emp.employeeData()

print("The Sum = ",add(100,200))

================================

Member Aliasing:

================

Syntax:

from module-name import member-name1 as new-name1, member-name2 as new-name2,...

from mod11 import x as var, add as Addition, product as Multiplier, employee as Employee

print(var)

print("The Sum = ",Addition(123,345))

print("The Product = ",Multiplier(12,23))

emp = Employee("Ashwini","Sr. Software Developer", 28,1600000)

emp.employeeData()

==============================================

dir():

=====

is an inbuilt method which we can use to get all the member names from the specified module.

Syntax:

dir()

==========================================

Math Module:

============

from math import \*

print(sqrt(97))

print(ceil(1.234)) # for the given float, the next highest integer value can be returned

print(floor(1.234)) # for the float, the before integer value (less than the given)

print(pow(9,7))

print(fabs(12))

print(fabs(-12))

print(log(10))

print(log2(10))

print(sin(23))

===========================================

Random Module:

==============

from random import \*

# random() ==> generate a random number from 0 to 1

print(random())

# randint() ==> generate a random integer from the specified range

print(randint(10,100))

# uniform() ==> to generate a float value from the specified range

print(uniform(1,10))

# randrange() ==> can generate an integer randomly from the specified range with difference/step

print(randrange(1,100))

print(randrange(1,100,10)) # 1, 11, 21,31,41,51,61,71,81,91

# choice() ==> can define with collections

# to generate the element from the collection randomly.

list1 = ["Apple", "Banana", "Papaya", "Mango", "Kiwi", "Jack-Fruit", "pineapple"]

print(choice(list1))

**Day53: Packages, File-Handling Day\_01:**

PACKAGES:

========

COLLECTION OF MODULES IS CALLED AS "PACKAGE".

PYCHARM ==> DIRECTORY ==> PYTHON FILE (MODULE)

DIRECTORY/FOLDER ==> PACKAGE

==> TREE TYPE OF HIERARCHY ==> PACKAGE

package ==> modules + \_\_init\_\_.py

=========================================

File Handling:

==============

Why files?

==========

to store the data permanently ==> files

==> Different types of files:

1) Text files

.txt

2) Binary Files

image files, audio files, video files etc.

How to open the file:

=====================

Syntax:

file-pointer = open("file-name.txt", "mode-name")

File modes:

===========

w-mode ==> writing into file

=============================

==> open the file with 'w' mode:

if the file is not available ==> 'w' ==> create the new file

if the file is available ==> open that existing file by deleting all the content.

r-mode ==> reading-mode

=======================

==> we can use to open the existed file for reading purpose

==> no modification while r-mode.

==> if the file is not existed ==> r-mode can give the error.

file not found error.

a-mode

======

==> a-mode can create the new file with specified name if it is not existed.

==> a-mode can also open the existing file for reading and writing both.

==> cannot overwrite the data in the file.

w+-mode

========

==> write + read

r+-mode

======

==> read + write

100-lines

writing is possible after 100-lines

a+-mode

=======

append + read

x-mode:

=======

exclusive mode

==> writing purpose

==>like the w-mode, it can create the new file for only writing

but it can give "file existing error", when we can use this to open the existing file.

# File Modes

fp = open("abcd.txt","w")

fp1 = open("pqrst.txt","r")

fp2 = open("xyz.txt","a")

fp3 = open("pqrst.txt","a")

fp4 = open("abc.txt","w+")

# fp5 = open("pqrst.txt","w+")

fp5 = open("pqrst.txt","r+")

# fp6 = open("pqrst.txt","x")

fp6 = open("ravi.txt","x")

**Day53: File handling Part\_02:**

File Handling:

==============

close the file:

===============

close():

=======

Syntax:

file-pointer.close()

Properties of File object/file pointer:

=======================================

1) getting of file name

========================

name: (property)

=====

file-object.name

2) getting of the file mode:

============================

mode: (property)

=====

file-object.mode

3) check whether the file is readable or not?

=============================================

readable()

===========

Syntax:

file-object.readable() ==> True/False

4) check whether the file is writeable or not?

==============================================

writeable()

===========

file-object.writeable() ==> True/False

5) is file closed?

==================

closed: (property)

======

file-pointer.closed ==> True/False

# file Object Properties

fp = open("ravi.txt","r")

print("Name of the file = ",fp.name)

print("Mode of the file = ",fp.mode)

print("Is file readable? = ",fp.readable())

print("Is file writeable? = ",fp.writable())

print("Is file closed? = ",fp.closed)

# closing the file

fp.close()

print("Is file closed? = ",fp.closed)

===============================================================

File Operations:

================

Writing the data into files:

============================

1) write():

==========

==> we can write string of data into a file (within the line).

Syntax:

file-object.write(string-data)

# writing operations

# fp = open("ravi.txt","w")

# fp1 = open("ravi.txt","w+")

# fp = open("a.txt","x")

# fp = open("a.txt","r")

fp = open("a.txt","a")

"""

to write data into file:

we can use: "w-mode", "w+-mode", "r+-mode", "a-mode", "a+-mode","x-mode"

"""

# writing of data into file when it is in w-mode

# fp.write("Python\t")

# fp.write("Object\t")

# fp.write("Oriented\t")

# fp.write("Programming\t")

# fp.write("Language")

fp.write("Hello")

# closing the file

fp.close()

=======================

writelines()

============

==> is used to write the data into multiple lines of the file.

Syntax:

file-object.writelines(list of lines of content)

# writing operation using writelines()

fp = open("c.txt","x")

ld = ["Python\n","Object\n","Oriented\n","Programming\n","Language\n"]

# fp.write(ld)

# fp.write(ld[0])

fp.writelines(ld)

fp.close()

=============================================

Reading of data from the file:

==============================

four possible ways:

1) read()

2) read(n)

3) readline()

4) readlines()

1) read():

=========

==> used to read the total data from the file.

Syntax:

file-pointer.read()

# reading with read()

fp = open("ravi.txt","r")

fp1 = open("a.txt","r+")

fp2 = open("c.txt","a+")

data = fp.read()

data1 = fp1.read()

# data2 = fp2.read()

print(data)

print(data1)

# print(data2)

fp.close()

fp1.close()

# fp2.close()

===================================

2) read(n):

===========

==> used to read the data from the file upto 'n' characters.

Syntax:

file-pointer.read(n)

fp = open("c.txt","r")

# data = fp.read(10)

data = fp.read(20)

print(data)

fp.close()

3) readline():

==============

used to read the data of only one line at a time.

Syntax:

file-object.realine()

fp = open("c.txt","r")

data = fp.read(10)

data = fp.read(20)

data1 = fp.readline()

data2 = fp.readline()

print(data)

print(data1)

print(data2)

for i in range(5):

fp.readline()

fp.close()

=======================================

readlines()

==========

==> to read all the lines of file ==> we can use "readlines()".

Syntax:

file-pointer.readlines()

fp = open("c.txt","r")

data = fp.readlines()

print(data)

for i in data:

print(i)

with statement:

===============

==> with is the keyword

==> while opening the file, with is used.

to define the block of operations on the file.

Syntax:

with open("file.txt","mode") as file-object:

block of operations

# with statement

with open("c.txt","a") as fp:

fp.write("Hi\n")

fp.write("Hello\n")

fp.write("Good Morning.")

print("Is file closed? = ",fp.closed)

print("Is file readable? = ",fp.readable())

fp.close()

print("Is file closed? = ",fp.closed)

**Day54: File handling Part\_03:**

File Handling:

==============

tell():

=======

==> an inbuilt method in python

which we can use to get the current location of the file-pointer/file-object from the beginning of the file.

Syntax:

file-pointer.tell()

# tell()

fp = open("abc.txt","r+")

print("The Position of fp = ",fp.tell())

fp.read(5)

print("The Position of fp = ",fp.tell())

fp.read(10)

print("The Position of fp = ",fp.tell())

fp.write("Continue.")

fp.readline()

print("The Position of fp = ",fp.tell())

===============================================

seek():

=======

==> is an inbuilt method

which we can use to move the file pointer from one position to another position.

Syntax:

file-object.seek(offset, from-where)

here:

offset ==> number of positions, that how many need to move

from-where ==> the start point to define the move

from-where:

0 ==> from the beginning of the file

1 ==> from the current position.

2 ==> from the end of the file.

Note:

====

python-3 can allowed to define the seek() for moving the file pointer from the beginning by default.

from-where with the values '1' and '2' ==> not supported in python-3.

# seek()

data = "Here is the Python Full Stack Content."

fp = open("abc.txt","w")

fp.write(data)

with open("abc.txt","r+") as fp:

text = fp.read()

print(text)

print("The Current Position of fp = ",fp.tell())

fp.seek(17)

print("The Current Position of fp = ",fp.tell())

fp.seek(0)

print("The Current Position of fp = ", fp.tell())

fp.seek(20)

print("The Current Position of fp = ", fp.tell())

============================================

Checking of the file whether it is exists or not?

=================================================

import os

f = input("Enter the file name:")

if os.path.isfile(f):

print("File is Existed.")

else:

print("File is not Existed.")

===========================================

Q: WAP TO CHECK WHETHER THE FILE IS EXISTED OR NOT. IF THE FILE IS AVAILABLE, THEN PRINT ITS CONTENT.

import os,sys

fname = input("Enter the file to search:")

if os.path.isfile(fname):

print("File is Existed:",fname)

fp = open(fname,"r")

else:

print("File Does not Exist:",fname)

sys.exit(0)

print("The Content of file is = ")

data = fp.read()

print(data)

fp.close()

========================================================

Assignment:

===========

WAP TO PRINT THE NUMBER OF LINES, WORDS AND CHARACTERS PRESENT IN THE FILE.

========================================

HANDLING OF BINARY FILES:

=========================

MODES FOR BINARY FILES TO HANDLE:

r, w, a, r+, w+, a+, x ==> Text file modes

rb, wb, ab, r+b, w+b, a+b, xb ==> Binary file modes

# WAP TO READ IMAGE FILE AND WRITE TO A NEW IMAGE FILE.

f1 = open("image1.jpg","rb")

f2 = open("image2.jpg","wb")

bytes = f1.read()

f2.write(bytes)

print("The New Image has created from Image1 Successfully.")

=====================================================

**Day55: Decorators Part\_01:**

DECORATORS:

===========

Decorator is a function that takes another function as an argument add some kinds of functionality without modifying the original functionality and returns a function.

Syntax:

def decorator(func):

add some functionality

return func

1) how the function can take a function as an argument:

=======================================================

Syntax:

def function-name(func):

implementation

# Sending a function as an argument to another function:

def do\_it\_again(func):

func()

func()

func()

def greets():

print("Hi")

print("Good Morning")

print("Welcome to Ashok IT.")

do\_it\_again(greets)

2) how the function can return a function

==========================================

Syntax:

def function-name(function(optional)):

implementation

return function

# Function returning a function:

def return\_upper():

return str.upper

to\_upper = return\_upper() # function-aliasing

"""

renaming the function

or

define the another name to the function which we have defined is called as "function aliasing".

"""

result = to\_upper("object oriented programming language")

print(result)

# print(to\_upper("object oriented programming language"))

3) inner function

=================

writing a function inside the another function.

Syntax:

def outer():

def inner(): # local function to outer()

implementation

inner()

inner()

# inner function

def parentFunction():

print("This is the implementation of parentFunction()") # 1

def childFunction1():

print("This is the first child's implementation.") # 3

def childFunction2():

print("This is the second child's implementation.") # 2

# childFunction1()

childFunction2()

childFunction1()

parentFunction()

# childFunction1()

==============================================

Creating of a Decorator:

========================

Syntax:

func = decorator(func)

# Creation of Decorator:

def decorator(function):

def wrapperFunction():

print("This is printed before the function is called.")

function()

print("This is printed after the function is called.")

return wrapperFunction

def greetings():

print("Hi")

print("Good Morning.")

print("Welcome to Ashok IT")

greetings = decorator(greetings)

greetings()

==================================================

Syntactic Decorator:

====================

@decorator:

def function(arg1, arg2, arg3,...):

pass

# Syntactic Decorator

def decorator(function):

def wrapperFunction():

print("This is printed before the function is called.")

function()

print("This is printed after the function is called.")

return wrapperFunction

@decorator

def greetings():

print("Hi")

print("Good Morning")

print("Welcome To Ashok IT.")

greetings()

**Day56: Decorators Part\_02:**

DECORATORS:

===========

Docstring:

=========

# Docstring

# used to define the description about any block (class, function, method etc.)

# during the debugging.

# can be defined with """ """

def fun1(t):

"""

This function is defined to print greetings

like: "good morning"

"good afternoon"

"good evening"

and "good night".

"""

if t >= 0 and t < 12:

print("Good Morning.")

elif t >= 12 and t < 16:

print("Good Afternoon.")

elif t >= 16 and t < 20:

print("Good Evening.")

else:

print("Good Night.")

fun1(6)

print(help(fun1))

print(help(print))

Preserving the Original Name and Docstring of the Decorator:

============================================================

def decorator(function):

def wrapperFunction():

print("This is printed before the function is called.")

function()

print("This is printed after the function is called.")

return wrapperFunction

@decorator

def greets():

"""

This function says hello when we called.

"""

print("Hello All.")

print(greets.\_\_name\_\_)

help(greets)

==> To preserve the identity of function and the documentation string of the function, we can use "decorator".

wraps() ==>

functools

import functools

def decorator(function):

@functools.wraps(function)

def wrapperFunction():

print("This is printed before the function is called.")

function()

print("This is printed after the function is called.")

return wrapperFunction

@decorator

def greets():

"""

This function says hello when we called.

"""

print("Hello All.")

print(greets.\_\_name\_\_)

help(greets)

======================================

Reusing Decorator:

==================

1) create a python file for defining decorator

ex: decorators.py

==================

import functools

def iterativeFunction(function):

@functools.wraps(function)

def wrapperFunction():

print("This is printed before the function is called.")

function()

print("This is printed after the function is called.")

return wrapperFunction

2) Import that decorator into application/another python file where we want to reuse it.

Syntax:

from decorator-module-name import decorator-name

from decorators import iterativeFunction

@iterativeFunction

def greets():

print("Hello All.")

@iterativeFunction

def welcome():

print("Welcome To Ashok IT.")

greets()

welcome()

===============================================

Decorators With Parameters:

===========================

wrapper(with parameters)

Syntax:

def wrapper(\*args, \*\*kwargs)

# Decorator with parameters

import functools

from Decortaors.Day\_02.decorators import iterativeFunction

def iterateFunction(function):

@functools.wraps(function)

def wrapper(\*args, \*\*kwargs):

function(\*args, \*\*kwargs)

function(\*args, \*\*kwargs)

return wrapper

@iterateFunction

def greets(name):

print("Hello",name)

# greets("Rakesh")

pname = "Suresh"

greets(name = pname)

=============================================

Returning values from Decorator Function:

=========================================

import functools

def decorator(function):

@functools.wraps(function)

def innerFunction(\*args, \*\*kwargs):

return function(\*args,\*\*kwargs)

return innerFunction

@decorator

def addition(n1,n2,n3):

print("The sum of {n1}, {n2} and {n3} is = ")

return n1 + n2 + n3

print("The sum is = ",addition(11,111,1111))

=====================================================

Do we define the Decorators with Arguments?:

============================================

Yes

It's all based on the inner function of the decorator.

If the inner function of the decorator with parameters ==> we can send arguments to decorator.

otherwise ==> we can't

========================================================

Chaining Decorators:

====================

Defining multiple decorators to a single function is called as "Chaining Decorator".

# Chaining Decorator

import functools

def splitString(function):

@functools.wraps(function)

def innerFunction(\*args,\*\*kwargs):

return function(\*args, \*\*kwargs).split()

return innerFunction

def toUpper(func):

@functools.wraps(func)

def wrapperFunction(\*args, \*\*kwargs):

return func(\*args,\*\*kwargs).upper()

return wrapperFunction

@splitString

@toUpper

def String(data):

return data

print(String("object oriented programming language"))

**Day57: Generators:**

Generators:

===========

What are generators?

=====================

==> generators are functions in python

==> used to generate/create iterators

==> iterators can perform traversing on list/tuple etc.

==> return a traversal object.

helps to traverse all the elements/items one at a time present in the iterator.

==> The difference between the normal function and generator function is:

1) the normal function can use "return" statement to return a value.

def normal\_function():

return 10,20,30,40

t1,t2,t3,t4 = normal\_function()

print(t1,t2,t3,t4)

2) the generator function can use "yield" statement to return values.

def gen\_fun():

yield 10

yield 20

yield 30

for i in gen\_fun():

print(i)

==================================

return Vs yield:

================

1) yield is the keyword, we can use in generator function.

return is the keyword, we can use in normal function.

2) yield is responsible for controlling the flow of the generator function. After returning the value using "yield", it pauses the execution by saving the states.

Whereas, the return statement returns the values and terminates the function.

=========================================================

Difference between Generator function and normal function:

==========================================================

next():

======

an inbuilt function

which can helps to understand whether the generator function is pause the implementation while returning the value.

def sequence(x):

for i in range(x):

yield i

range\_values = sequence(10)

print(next(range\_values))

print(next(range\_values))

print(next(range\_values))

print(next(range\_values))

print(next(range\_values))

print(next(range\_values))

print(next(range\_values))

print(next(range\_values))

print(next(range\_values))

print(next(range\_values))

# print(next(range\_values))

=======================================

Defining the generator function using generator expression:

============================================================

lambda function ==> anonymous function

p = 7

genr = (i for i in range(p) if i % 2 == 0) # generator object

list\_values = [i for i in range(p) if i % 2 == 0] # list variable

print(genr) # we can't access the generator object directly.

print(list\_values)

for k in genr:

print(k)

=================================================

Uses of Generators:

===================

1) Easy to Implement

2) Memory Efficient

3) Infinite Sequence

**Day58: Exception Handling:**

Exception Handling:

===================

two types of errors:

1) Syntax Errors

2) Runtime Errors

1) Syntax Error:

================

The error which occurs because of invalid syntax.

# Syntax Errors

# program to find the number is even or odd

num = 27

if num % 2 == 0

print("It is an Even number.")

else:

print("It is an Odd number.")

Note:

=====

Syntax Errors ==> Human Mistakes/Developer mistakes

==> responsible by human

Run Time Error:

===============

==> Also called as an Exceptions.

==> Runtime errors, we can observer during the execution of the program

==> Because of:

1) User input

2) Program logic

3) memory issue etc.

a = int(input("Enter a value:"))

b = 0

print(type(a))

print(a/b)

What is an Exception?

=====================

An unwanted and unexpected even that disturbs the normal flow of the program execution is called as an "Exception".

==> For exception, there is a class.

Ex: Value Error,

Zero Division Error,

Type Error,

Index Error etc.

Exception Handling:

===================

Exception will stop the program

or

will disturb the normal flow of execution of the program

==> to make execute the program normally by handling an exception ==> Exception handling is used.

try-except block ==> to handle the exceptions

Syntax:

try:

program that we want to execute normally

even with exception

except:

exception

# Exception Handling

a = int(input("Enter a value:"))

b = int(input("Enter a value:"))

try:

res = a / b

except ZeroDivisionError:

res = 0

print("Dividing the number with 0 is not possible.")

print(res)

===============================

Try with Multiple Exceptions:

===============================

# evens = [2,4,6,8] # with total four index ==> 0 to 3

#

# print(evens[4])

try:

evens = [2,4,6,8]

print(evens[4])

except ZeroDivisionError:

print("Dividing a number with 0 is not possible.")

except IndexError:

print("Index out of range is used.")

except TypeError:

print("Index is not supported to use.")

===============================

try-except-else block:

=======================

When we might want to execute/run a certain block of code if the code block inside the try runs without any errors.

In this, we can use "else".

try:

num = int(input("Enter a value:"))

assert num % 2 == 0

except:

print("Not an Even number.")

else:

rec = 1/num

print(rec)

================================

Semester Results:

=================

JNTU ==>

person ==> 4/6

sub-1:60

sub-2: 40

sub-3: 50

sub-4: 70

sub-5: A

sub-6: A

Phonepe:

=======

sending amount ==> friend

5000

Failure ==> Server Issue

Technical issue at bank

bank server running server

**Day59: DSA Part\_01:**

Data Structures and Algorithms:

===============================

Strings and manipulations

Lists

Tuple

Sets

Dictionary

Frozen Sets

Bytes

Byte arrays

Collection Module:

==================

==> an inbuilt module in python

==> provides an alternative to all inbuilt python datatypes like: list, tuple etc.

==> collection module contains lot of built-in datatypes which are alternatives to above datatypes.

1) counters:

=============

==> subclass of dictionary designed to count the hash able object.

ex: ["apple", "orange", "banana", "apple", "mango", "orange", "papaya", "banana", "apple", "kiwi"]

==> "apple" ==> 3

"orange" ==> 2

"banana" ==> 3

"mango" ==> 1

"papaya" ==>

"kiwi" ==> 1

# counters

from collections import Counter

# create the counter

# Counter()

data = Counter(['apple', 'banana', 'orange', 'apple', 'papaya', 'orange', 'mango', 'papaya', 'kiwi','apple'])

print("The Given Counter is :")

print(data)

# update the counter

# update()

data.update(['strawberry','grapes'])

print(data)

# Accessing the elements from the counter

# most\_common()

print(data.most\_common(1))

print(data.most\_common(0))

print(data.most\_common(5))

# print(data[0])

==============================================

OrderedDict:

============

==> a sub-class of the dictionary that maintains the sequence of items as we were added.

==> gives the facility to add/insert and remove elements.

# orderedDict

from collections import OrderedDict

# create the orderedDict

#OrderedDict()

data = OrderedDict([('a',100),('b',200),('c',300),('d',400),('e',500)])

print("The Given Ordered Dictionary is = ")

print(data)

# Insertion of elements into ordered dictionary

# update()

data.update({'f' : 600})

# data.update((('g',700)))

data.update([('g',700)])

data.update({('h',800)})

print(data)

# Reversing the order

# reversed()

res\_data = OrderedDict(reversed(list(data.items())))

print("The Reversed Ordered Dictionary = ")

print(res\_data)

# print(reversed(data))

==============================================

DefaultDict:

============

==> a sub-class of dictionary

that calls a factory function to supply missing values, simplifying handling of missing keys.

# Default Dict

from collections import defaultdict

# create the default dict

# data = defaultdict(int)

# data = defaultdict(float)

data = defaultdict(str)

print(data)

data['a'] = '12.2'

data['b'] += '2.34'

print(data)

==========================================

ChainMap:

=========

==> groups multiple dictionaries into a single view by making it convenient to manage with multiple scopes.

from collections import ChainMap

d1 = {'a' : 111, 'e' : 222, 'i' : 333, 'o' : 444, 'u' : 555}

d2 = {'b' : 100, 'a' : 200, 'p' : 300, 'e' : 400}

print(d1)

print(d2)

result = ChainMap(d1, d2)

print(result)

Namedtuple:

===========

# Named Tuple

from collections import namedtuple

# create the named tuple

data = namedtuple('data',['name','quantity'])

p1 = data(name = 'a',quantity=121)

print(data)

print(p1)

print(p1[1])

# print(p1['a'])

==============================================

Linked List:

============

==> a linear collection of data elements where each element pointes to the next, allowing for a dynamic data structure with efficient insertion and deletions.

# Linked List

class Node:

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

self.data = data

self.next = None

class LinkedList:

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

self.head = None

def printList(self):

temp = self.head

while temp:

print(temp.data)

temp = temp.next

lld = LinkedList()

lld.head = Node(100)

s = Node(200)

t = Node(300)

f = Node(400)

lld.head.next = s

s.next = t

t.next = f

lld.printList()

**Day60: DSA Part\_02:**

Stack:

======

==> a linear structure

going to work with the principle "LIFO (Last In and First Out)

# Stack

class Stack:

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

self.items = list() # added an empty list

def push(self, element):

# appending of elements into an empty list

self.items.append(element)

def pop(self):

return self.items.pop()

def peek(self):

return self.items[-1] # rightmost element in the list (Last element of list)

def is\_empty(self):

return self.items == list()

s = Stack()

print("The Given List = ",s.items)

# adding of element to the list

s.push('python ')

s.push('programming ')

s.push('language')

print(s.items)

print(s.peek())

ret\_obj = s.pop()

print("The Return Object = ",ret\_obj)

print(s.peek())

Queue:

======

==> A linear structure like stack

==> based on principle "FIFO (First In and First Out)"

Implementing the Queue using List:

==================================

# Queue with List data

Queue = list() # empty list

Queue.append(10)

Queue.append(20)

Queue.append(30)

Queue.append(40)

Queue.append(50)

print("The Queue Before the Deque Operation is = ")

print(Queue)

# Deque Operation (Removing the first element) [10,20,30,40,50]

ret1 = Queue.pop(0) # First In [20,30,40,50]

ret2 = Queue.pop(0) # [30,40,50]

ret3 = Queue.pop(0) # [40,50]

ret4 = Queue.pop(0) # [50]

ret5 = Queue.pop(0) # []

print(ret1)

print(ret2)

print(ret3)

print(ret4)

print(ret5)

print("The Queue after the DeQue Operation is = ")

print(Queue)

==========================================

Deque:

=====

Double Ended Queue

is a generalization of stacks and Queues.

Which it supports:

memory Efficiency

fast append

pops elements from either side.

pop() ==> remove/pop from last

pop(0) ==> remove/pop from beginning/start

# Deque

from collections import deque

# Create a deque

d = deque(['a', 'e', 'i', 'o', 'u'])

print("The Given Deque = ",d)

# Appending to the Deque at the right

d.append('A')

d.append('E')

d.append('I')

d.append('O')

d.append('U')

print("After the Appending Operation the Deque is = ")

print(d)

# pop from left

# popleft() ==> it can pop the left most element

d.popleft()

print("The Deque After the left pop operation is = ")

print(d)

# pop from right

# pop() ==> it can pop the right most element

d.pop()

print("The Deque After the right pop operation is = ")

print(d)

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

# ld.popleft()

=============================================

Queue Implementation with collections.deque:

============================================

# Queue with Deque

from collections import deque

queue = deque() # empty queue

print("The Queue without Enque = ",queue)

#Enque the elements into the above definition

queue.append('a')

queue.append('e')

queue.append('i')

queue.append('o')

queue.append('u')

print("The Given Queue = ",queue)

# Deque from left

print(queue.popleft())

print("The Queue after the left pop operation is = ",queue)

# Deque from right

print(queue.pop())

print("The Queue after the right pop operation is = ",queue)

============================================

Queue Implementation with Queue()

=================================

from queue import Queue

q = Queue(maxsize=7)

print("The Length of the Queue = ",q.qsize())

# Enque Operation

# q.append(10)

q.put(10)

q.put(20)

q.put(30)

print("The Length of the Queue = ",q.qsize())

q.put(40)

q.put(50)

q.put(60)

q.put(70)

print("The Length of the Queue = ",q.qsize())

# q.put(1234)

#

# print("The Length of the Queue = ",q.qsize())

# Deque Operation

# q.pop()

res1 = q.get()

res2 = q.get()

print(res1)

print(res2)

print(q.get())

print(q.get())

print(q.get())

print(q.get())

print(q.get())

# print(q)

# is queue is empty?

print(q.empty())

**Day61: DSA Part\_03:**

Priority Queue:

===============

list ==> [1,2,3,4,5]

iteration:

order/sequence

priority Queue => [1,3,5,7,9]

each element ==> priority

7 ==> high

9 ==> second

3 ==> third

1 ==> fourth

5 ==> last

# Priority Queue

class priorityQueue:

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

self.queue = [] # empty priority Queue

def \_\_str\_\_(self):

return ' '.join([str(i) for i in self.queue])

def isEmpty(self):

return len(self.queue) == []

def insert(self,item):

self.queue.append(item)

def delete(self):

try:

highest\_priority = 0

for i in range(len(self.queue)):

if self.queue[i] > self.queue[highest\_priority]:

highest\_priority = i

item = self.queue[highest\_priority]

del self.queue[highest\_priority]

return item

except IndexError:

print("This Queue is Empty.")

return None

pq = priorityQueue()

pq.insert(400)

pq.insert(300)

pq.insert(500)

pq.insert(700)

pq.insert(900)

print(pq.queue)

print(pq.isEmpty())

val1 = pq.delete()

print(val1)

print(pq.delete())

print(pq.delete())

print(pq.delete())

print(pq.delete())

print(pq.delete())

=======================================

Heap Queue:

===========

# heap

import heapq

from heapq import heapify, heappush

# defining the heap

heap\_data = [1000,123,5797,7997,97,1001,1023,1225] # list

# Transform the list into heapqueue

heapq.heapify(heap\_data) # 7997 5797 2001 1993 1225 1073 1023 1001 1000 123 97

print(heap\_data)

# Adding elements into heapqueue

heapq.heappush(heap\_data,1073)

heapq.heappush(heap\_data,1993)

heapq.heappush(heap\_data,2001)

print(heap\_data)

pop\_element = heapq.heappop(heap\_data)

print(pop\_element)

========================================================================================

**Day62: DSA Part\_04:**

Binary Tree:

============

==> Tree represents the nodes connected by edges.

==> non-linear data structure.

==> properties:

1) one node among all must be a root.

2) each node except root node must be associated with parent node.

3) each node should contains two arbitrary number of child nodes (2).

# Binary tree

Creation of node as root:

=========================

class Node:

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

self.left = None

self.right = None

self.data = data

def printTree(self):

print(self.data)

print(self.left)

print(self.right)

root = Node(1993)

root.left = 2005

root.right = 2013

root.printTree()

Inserting into Tree:

====================

# Inserting to tree

class Node:

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

self.left = None

self.right = None

self.data = data # 100

def insert(self,data):

# comparing the new value with parent data

if self.data:

if data < self.data:

if self.left is None:

self.left = Node(data)

else:

self.left.insert(data)

elif data > self.data:

if self.right is None:

self.right = Node(data)

else:

self.right.insert(data)

else:

self.data = data

def printTree(self):

if self.left:

self.left.printTree()

print(self.data),

if self.right:

self.right.printTree()

root = Node(112) # parent data

root.insert(102) # 102 < 112 ==> True ==> left = 102

root.insert(144) # 144 > 112 ==> True ==> right = 144

root.insert(97) # 112 replace with 97

root.printTree()

======================================================

Traversing on Tree:

===================

three ways:

1) in-order Traversal ==> left, root and right

================================================

# Inorder Traversal

class Node:

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

self.left = None

self.right = None

self.data = data

def insert(self,data):

# comparing the new value with parent data

if self.data:

if data < self.data:

if self.left is None:

self.left = Node(data)

else:

self.left.insert(data)

elif data > self.data:

if self.right is None:

self.right = Node(data)

else:

self.right.insert(data)

else:

self.data = data

def printTree(self):

if self.left:

self.left.printTree()

print(self.data),

if self.right:

self.right.printTree()

# Inorder Traversal

# left ==> root ==> right

def inorderTraversal(self, root):

res = []

if root:

res = self.inorderTraversal(root.left)

res.append(root.data)

res = res + self.inorderTraversal(root.right)

return res

root = Node(27)

root.insert(14)

root.insert(35)

root.insert(10)

root.insert(19)

root.insert(31)

root.insert(42)

result = root.inorderTraversal(root)

print(result)

=============================================

2) pre-order Traversal ==> root , left, right

================================================================

# Pre-order traversal

class Node:

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

self.left = None

self.right = None

self.data = data

def insert(self,data):

# comparing the new value with parent data

if self.data:

if data < self.data:

if self.left is None:

self.left = Node(data)

else:

self.left.insert(data)

elif data > self.data:

if self.right is None:

self.right = Node(data)

else:

self.right.insert(data)

else:

self.data = data

def printTree(self):

if self.left:

self.left.printTree()

print(self.data),

if self.right:

self.right.printTree()

# pre-order traversal

# root ==> left ==> right

def preorderTraversal(self, root):

res = []

if root:

res.append(root.data) # root visiting

res = res + self.preorderTraversal(root.left) # left visiting

res = res + self.preorderTraversal(root.right) # right visiting

return res

root = Node(14) # root

root.insert(27) # right

root.insert(10) # left

root.insert(35) # right

root.insert(19) # right

root.insert(42)

root.insert(31)

print(root.preorderTraversal(root))

==================================================================

3) Post-order Traversal ==> left ==> right ==> root

# Post Order Traversal

class Node:

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

self.left = None

self.right = None

self.data = data

def insert(self,data):

# comparing the new value with parent data

if self.data:

if data < self.data:

if self.left is None:

self.left = Node(data)

else:

self.left.insert(data)

elif data > self.data:

if self.right is None:

self.right = Node(data)

else:

self.right.insert(data)

else:

self.data = data

def printTree(self):

if self.left:

self.left.printTree()

print(self.data),

if self.right:

self.right.printTree()

# post order Traversal

# left ==> right ==> root

def postOrderTraversal(self,root):

res = []

if root:

res = res + self.postOrderTraversal(root.left) # left visiting

res = res + self.postOrderTraversal(root.right) # right visiting

res.append(root.data) # root visiting

return res

root = Node(27) # root

root.insert(14) # left

root.insert(35) # right

root.insert(10) # left

root.insert(19) # right

root.insert(31) # left

root.insert(42) # right

print(root.postOrderTraversal(root))

**Day63: DSA Part\_05:**

Graphs:

=======

it is a pictorial representation of a set of objects where some pair of objects are connected by the links.

==> The interconnection between the objects are represented as "vertices".

==> links that connects to the vertices are called as "edges".

# graph

graph = {

"a" : ["b","c"],

"b" : ["a","d"],

"c" : ["a","d"],

"d" : ["b","e"],

"e" : ["d"]

}

print(graph)

====================

Displaying of graph Vertices:

=============================

class graph:

def \_\_init\_\_(self,gdict = None):

if gdict is None:

gdict = []

self.gdict = gdict

def getVertices(self):

return list(self.gdict.keys())

graph\_structure = {

"a" : ["b","c"],

"b" : ["a","d"],

"c" : ["a","d"],

"d" : ["b","e"],

"e" : ["d"]

}

gobj = graph(graph\_structure)

print("The Vertices of the given graph are:",gobj.getVertices())

Displaying graph edges:

=======================

class graph:

def \_\_init\_\_(self,gdict = None):

if gdict is None:

gdict = []

self.gdict = gdict

def edges(self):

return self.findedges()

def findedges(self):

edgename = []

for i in self.gdict: # keys vertices

for j in self.gdict[i]: # values edges

if(j,i) not in edgename:

edgename.append({i,j})

return edgename

graph\_structure = {

"a" : ["b","c"],

"b" : ["a","d"],

"c" : ["a","d"],

"d" : ["b","e"],

"e" : ["d"]

}

gobj = graph(graph\_structure)

print("The Edges = ",gobj.edges())

==================================================

Adding Vertex:

==============

class graph:

def \_\_init\_\_(self,gdict = None):

if gdict is None:

gdict = []

self.gdict = gdict

def getVertices(self):

return list(self.gdict.keys())

def addVertex(self,vertex):

if vertex not in self.gdict:

self.gdict[vertex] = []

graph\_data = {

"a" : ["b","c"],

"b" : ["a","d"],

"c" : ["d","a"],

"d" : ["b","e"],

"e" : ["d"]

}

gobj = graph(graph\_data)

gobj.addVertex("f")

print(gobj.getVertices())

===============================================

Adding of an edge to the graph:

===============================

class graph:

def \_\_init\_\_(self,gdict = None):

if gdict is None:

gdict = []

self.gdict = gdict

def edges(self):

return self.findEdges()

def findEdges(self):

edgename = []

for i in self.gdict: # keys vertices

for j in self.gdict[i]: # values edges

if (j, i) not in edgename:

edgename.append({i, j})

return edgename

def addEdge(self,edge):

edge = set(edge)

(vertex1,vertex2) = tuple(edge)

if vertex1 in self.gdict:

self.gdict[vertex1].append(vertex2)

else:

self.gdict[vertex1] = [vertex2]

graph\_data = {

"a" : ["b","c"],

"b" : ["a","d"],

"c" : ["a","d"],

"d" : ["b","e"],

"e" : ["d"]

}

gobj = graph(graph\_data)

gobj.addEdge({'a','e'})

gobj.addEdge({'a','c'})

print(gobj.edges())

=================================================

Searching Algorithms:

=====================

Ecoomerce: Flipkart

clothes:Kart(02-10-10pm)

phones: Kart(02-10-8pm)

beauty products:Kart(03-10-6am)

1) Linear search:

=================

# Linear search

def linearSearch(array,n,x):

for i in range(0,n):

if array[i] == x:

return i

return -1

array = [2,4,0,1,9,7,11,5]

x = 100

n = len(array)

resultant\_position = linearSearch(array,n,x)

if resultant\_position == -1:

print("The Given Searching object is not found in an array.")

else:

print("The Given Searching object is identifies in an array at:{}".format(resultant\_position))

Binary Search:

==============

# Binary Search

def binarySearch(array, x, low, high):

while low <= high:

mid = low + (high-low)//2

if x == array[mid]:

return mid

elif x > array[mid]:

low = mid + 1

else:

high = mid - 1

return -1

array = [3,4,5,6,7,8,9]

x = 8

position = binarySearch(array,x,0,len(array)-1)

if position == -1:

print("Results not found.")

else:

print("The Searching object is at:",position)

**Day64: Linked List Practice\_01:**

# WAP IN PYTHON TO CREATE THE LINKED LIST

# HOW WE CAN CREATE THE LINKEDLIST IN PYTHON

====================================================

# Creation of linkedlist

class Node:

# general structure of node in a linkedlist

def \_\_init\_\_(self,data = None):

self.data = data

self.next = None

class singleLinkedList:

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

self.head = None

l1 = singleLinkedList()

l1.head = Node("Monday")

l2 = Node("Tuesday")

l3 = Node("Wednesday")

l4 = Node("Thursday")

l5 = Node("Friday")

l6 = Node("Saturday")

l7 = Node("Sunday")

l1.head.next = l2

l2.next = l3

l3.next = l4

l4.next = l5

l5.next = l6

l6.next = l7

==============================================

# WAP in python to print the data of linked list

================================================

# WAP to traverse on the linked list in Python

===============================================

# Printing of linkedlist data

# Creation of linkedlist

class Node:

# general structure of node in a linkedlist

def \_\_init\_\_(self,data = None):

self.data = data

self.next = None

class singleLinkedList:

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

self.head = None

def printLinkedList(self):

value = self.head

# Traversing on the linkedlist

while value is not None:

print(value.data)

value = value.next

l1 = singleLinkedList()

l1.head = Node("Monday")

l2 = Node("Tuesday")

l3 = Node("Wednesday")

l4 = Node("Thursday")

l5 = Node("Friday")

l6 = Node("Saturday")

l7 = Node("Sunday")

l1.head.next = l2

l2.next = l3

l3.next = l4

l4.next = l5

l5.next = l6

l6.next = l7

l1.printLinkedList()

==================================================

# WAP to insert the data into linked list

==========================================

**Day65: DSA Practice With LinkedList:**

Insert the new node to the linked list at the beginning:

========================================================

# Printing of linkedlist data

# Creation of linkedlist

class Node:

# general structure of node in a linkedlist

def \_\_init\_\_(self,data = None):

self.data = data

self.next = None

class singleLinkedList:

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

self.head = None

def printLinkedList(self):

value = self.head

# Traversing on the linkedlist

while value is not None:

print(value.data)

value = value.next

# adding new node at the beginning of linkedlist

def atBegin(self,ndata):

nnode = Node(ndata)

nnode.next = self.head

self.head = nnode

l1 = singleLinkedList()

l1.head = Node("Monday")

l2 = Node("Tuesday")

l3 = Node("Wednesday")

l4 = Node("Thursday")

l5 = Node("Friday")

l6 = Node("Saturday")

l7 = Node("Sunday")

l1.head.next = l2

l2.next = l3

l3.next = l4

l4.next = l5

l5.next = l6

l6.next = l7

# l1.printLinkedList()

l1.atBegin("Apple")

l1.printLinkedList()

===========================================================

How to insert the new node at the end of the linked list

========================================================

# Printing of linkedlist data

# Creation of linkedlist

class Node:

# general structure of node in a linkedlist

def \_\_init\_\_(self,data = None):

self.data = data

self.next = None

class singleLinkedList:

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

self.head = None

def printLinkedList(self):

value = self.head

# Traversing on the linkedlist

while value is not None:

print(value.data)

value = value.next

# adding new node at the beginning of linkedlist

def atBegin(self,ndata):

nnode = Node(ndata)

nnode.next = self.head

self.head = nnode

# adding new node at the end of linkedlist

def atEnd(self,ndata):

nnode = Node(ndata)

if self.head is None:

self.head = nnode

return

last = self.head

while last.next:

last = last.next

last.next = nnode

l1 = singleLinkedList()

l1.head = Node("Monday")

l2 = Node("Tuesday")

l3 = Node("Wednesday")

l4 = Node("Thursday")

l5 = Node("Friday")

l6 = Node("Saturday")

l7 = Node("Sunday")

l1.head.next = l2

l2.next = l3

l3.next = l4

l4.next = l5

l5.next = l6

l6.next = l7

# l1.printLinkedList()

l1.atBegin("Apple")

l1.atEnd("Mango")

l1.printLinkedList()

=======================================================

How to insert the node between two nodes:

=========================================

# Printing of linkedlist data

# Creation of linkedlist

class Node:

# general structure of node in a linkedlist

def \_\_init\_\_(self,data = None):

self.data = data

self.next = None

class singleLinkedList:

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

self.head = None

def printLinkedList(self):

value = self.head

# Traversing on the linkedlist

while value is not None:

print(value.data)

value = value.next

# adding new node at the beginning of linkedlist

def atBegin(self,ndata):

nnode = Node(ndata)

nnode.next = self.head

self.head = nnode

# adding new node at the end of linkedlist

def atEnd(self,ndata):

nnode = Node(ndata)

if self.head is None:

self.head = nnode

return

last = self.head

while last.next:

last = last.next

last.next = nnode

# Adding new node between two nodes

def inBetween(self,m\_node,ndata):

if m\_node is None:

print("The Mentioned node is not in a linkedlist.")

return

nnode = Node(ndata)

nnode.next = m\_node.next

m\_node.next = nnode

l1 = singleLinkedList()

l1.head = Node("Monday")

l2 = Node("Tuesday")

l3 = Node("Wednesday")

l4 = Node("Thursday")

l5 = Node("Friday")

l6 = Node("Saturday")

l7 = Node("Sunday")

l1.head.next = l2

l2.next = l3

l3.next = l4

l4.next = l5

l5.next = l6

l6.next = l7

# l1.printLinkedList()

l1.atBegin("Apple")

l1.atEnd("Mango")

l1.inBetween(l6.next,"Python")

l1.printLinkedList()

==============================================

Removing of node from linked list:

==================================

# Printing of linkedlist data

# Creation of linkedlist

class Node:

# general structure of node in a linkedlist

def \_\_init\_\_(self,data = None):

self.data = data

self.next = None

class singleLinkedList:

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

self.head = None

def printLinkedList(self):

value = self.head

# Traversing on the linkedlist

while value is not None:

print(value.data)

value = value.next

# adding new node at the beginning of linkedlist

def atBegin(self,ndata):

nnode = Node(ndata)

nnode.next = self.head

self.head = nnode

# adding new node at the end of linkedlist

def atEnd(self,ndata):

nnode = Node(ndata)

if self.head is None:

self.head = nnode

return

last = self.head

while last.next:

last = last.next

last.next = nnode

# Adding new node between two nodes

def inBetween(self,m\_node,ndata):

if m\_node is None:

print("The Mentioned node is not in a linkedlist.")

return

nnode = Node(ndata)

nnode.next = m\_node.next

m\_node.next = nnode

# remove the node

def removeNode(self,key):

h = self.head

if h is not None:

if h.data == key:

self.head = h.next

h = None

return

while h is not None:

if h.data == key:

break

preview = h

h = h.next

if h == None:

return

preview.next = h.next

h = None

l1 = singleLinkedList()

l1.head = Node("Monday")

l2 = Node("Tuesday")

l3 = Node("Wednesday")

l4 = Node("Thursday")

l5 = Node("Friday")

l6 = Node("Saturday")

l7 = Node("Sunday")

l1.head.next = l2

l2.next = l3

l3.next = l4

l4.next = l5

l5.next = l6

l6.next = l7

# l1.printLinkedList()

l1.atBegin("Apple")

l1.atEnd("Mango")

l1.inBetween(l6.next,"Python")

l1.removeNode("Friday")

l1.printLinkedList()

============================================

# WAP TO SORT THE COLLECTION/DATA STRUCTURE WITH BUILT-IN METHODS

# [1,0,7,9,2,5]

# [0,1,7,9,2,5]

# [0,1,7,2,9,5]

# [0,1,7,2,5,9]

# [0,1,2,7,5,9]

# [0,1,2,5,7,9]

def Sorting(ld):

for i in range(len(ld)-1,0,-1):# 5 2 9 7 0 1

for j in range(i): # 0 to 4

if ld[j] > ld[j+1]:

temp = ld[j]

ld[j] = ld[j+1]

ld[j+1] = temp

ld = [19,2,31,45,6,11,121,27]

print(ld)

Sorting(ld)

print(ld)

**Day66: Introduction to Advanced Python:**

CORE PYTHON:

ALL THE PROGRAMMING FUNDAMENTALS

IDENTIFIERS

KEYWORDS

VARIABLES

IO OPERATIONS

OPERATORS

CONTROL STATEMENTS

DATA STRUCTURES

STRING MANIPULATIONS

LIST MANIPULATIONS

TUPLE MANIPULATIONS

SET MANIPULATIONS

DICTIONARY MANIPULATIONS

OTHER DATA STRUCTURES

FROZENSETS

BYTES

BYTE ARRAY

FUNCTIONS

OBJECT ORIENTED PROGRAMMING

CLASS, OBJECT, METHODS, CONSTRUCTOR, GARBAGE COLLECTION

ENCAPSULATION, INHERITANCE, POLYMORPHISM ETC.

FILE HANDLING

EXCEPTION HANDLING

ADVANCED PYTHON

===============

REGULAR EXPRESSIONS

DATABASE PROGRAMMING

PYTHON LIBRARIES

WEB DEVELOPER

GUI (GRAPHICAL USER INTERFACE) ==> TKINTER

DATASCIENCE, AI ==> NUMPY, PANDAS

PRACTICAL

DJANGO ==> FRAMEWORK

SOCKET PROGRAMMING

DOCKER, KUBERNETES, MAVEN

GITHUB

JIRA

=============================

DATA STRUCTURES

ARRAYS

LIST ==> SIMILAR TO ARRAY

ARRAY ==> COLLECTION OF SIMILAR DATA ITEMS

INT ARRAY ==> ALL INTEGERS ==> {1,2,3,4,5}

FLOAT ARRAY ==> ALL FLOATS ETC. {1.2,0.001,0.23}

import numpy

ecommerce ==> cart ==> collection of any item

we can add any element

{gadgets, clothes, ..}

from NumPy:

array definition

single dimension

multi-dimension

math operations

searching algorithms

sorting algorithms

insertion etc.

**Day67:**  **Introduction To Numpy:**

NumPy:

=====

NumPy ==> Numerical Python

why?

====

for performing the complex math operations.

open source and free software

we can use to perform array based computing.

array ==> collection of similar data items.

1-d array ==> one dimensional array

arranging of elements in either row or in column.

2-d array ==> two dimensional array

arranging of elements in rows and columns both. (matrix)

n-d array ==> n-dimensional array (Image processing)

History:

========

==> Developed by "Travis Oliphant"

==> in 2005

==> using C and Python.

C ==> fastest execution

Python ==> easy development

==> Travis gone through a module/library ==> Numeric Library (Jim Hungunian)

Python ==> Interpreter dependent language

C, C++, Perl and Shell script

==> NumPy ==> 3rd party module/package/library

we required to import this module

Syntax:

import numpy

==> we must require to install the numpy extensively before the importing.

==> to install any 3rd party modules:

in python only there is one package manager ==> pip

Syntax: (for installation)

pip install module-name

ex: pip install numpy

Syntax (for uninstallation)

pip uninstall module-name

ex: pip uninstall numpy

do we have pip?

===============

yes.

# WAP IN PYTHON TO PRINT THE VERSION OF NUMPY.

import numpy

print(numpy.\_\_version\_\_)

============================

==> we can use the numpy with alias name.

Syntax:

import numpy as alias-name

# WAP IN PYTHON TO GET VERSION OF NUMPY.

# numpy with alias name

import numpy as rk

print(rk.\_\_version\_\_)

**Day68:**  **Python List Vs NumPy Ndarray:**

Why NumPy?

==========

NumPy is a third-party module/package/library

which we can to use to perform the complex math operations

with array computing.

Array:

=====

It is a collection/sequence of similar data items (homogeneous elements).

ex: [1,2,3,4], {1.2,2.3,3.4},(9-7j,9+7j,7-5j,7+5j) etc.

Q: In Python, how we can store/represent with the similar data items?

=====================================================================

List ==> Homogeneous elements/Heterogeneous elements

Mutable

Insertion order can be preserved

Tuple ==> Homogeneous elements/Heterogeneous elements

Immutable

Insertion Order can be preserved

Set ==> Homogeneous elements/Heterogeneous elements

Mutable (add()) and/or Immutable

Cannot preserve the order.

Python List Vs Numpy:

=====================

In Python:

list of built-in datatypes:

int

float

complex

Bool

str

list

tuple

set

frozenset

bytes

bytearray

None

In NumPy:

we have only one built-in datatype

which we can use to define:

1D array ==> Arranging of elements in either row or column format.

2D array ==> Arranging of elements in both row and column

ND Array ==> Multi-dimensional array

i.e., "ndarray" is also a class

ndarray is used to store homogeneous elements only.

IDEs available for NumPy as default:

====================================

Jupiter

Anaconda Jupiter notebook

List Vs ndarray:

================

List data elements cannot reserve the continuous memory in heap section.

list to array:

==============

array():

we can convert the list data into an array

Syntax:

import numpy

numpy.array(list-data)

Getting the size of the memory:

===============================

getsizeof()

===========

==> a built-in function

used to get the size of the memory for any object.

==> defined in the module: "sys"

==> to use this "getsizeof()", we must be import "sys" module.

Syntax:

import sys

sys.getsizeof(object-name)

import sys

import numpy

ld = [1,2,3,4,5,6,7,8,9,10,11,12]

print(ld,type(ld),id(ld))

print("The Memory Size of given list data is = ",sys.getsizeof(ld))

print(id(ld[0]),id(ld[1]),id(ld[2]))

# converting the list into an array

a = numpy.array(ld)

print(a,type(a),id(a))

print("The Memory size of given array is = ",sys.getsizeof(a))

print(id(a[0]),id(a[1]),id(a[2]))

**Day69: Array Creation:**

Why NumPy?

==========

1) Python List ==> reserve with non-contiguous memory.

NumPy-ndarray ==> reserve with contiguous memory.

2) Python-List will consume more time and performance get disturbed.

NumPy-ndarray will consume less time and performance is good.

3) Python: 10+ datatypes ==> Heterogeneous datatypes

NumPy: 1-datatype ==> ndarray

import sys

import numpy

a = 123

b = 1.23

c = 1-2j

d = True

e = "python"

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

g = (2,3,4,5,6)

arr = numpy.array(f)

arr1 = numpy.array([1.2,2.3,3.4,4.5,5.6])

print("The Size of all the above data definitions are = ")

print(sys.getsizeof(a))

print(sys.getsizeof(b))

print(sys.getsizeof(c))

print(sys.getsizeof(d))

print(sys.getsizeof(e))

print(sys.getsizeof(f))

print(sys.getsizeof(g))

print(sys.getsizeof(arr))

print(sys.getsizeof(arr1))

===============================================================

ndarray:

========

built-in datatype in NumPy

a pre-defined class: ndarray

ndarray

======

==> most important object in NumPy.

==> N-dimensional Array Type

==> describe a collection of items of the same type

==> which can be accessed using a zero-based index.

dtype

=====

==> Each item of ndarray takes the same size of a memory block.

==> datatype object

array Scalar type

=================

==> any item can be extracted from an ndarray by using slicing is represented by a python object of an array called as "array scalar type".

=============================

How to create an ndarray?

=========================

[1,2,3] ==> 1d-array ==> row

[1

2

3] ==

; 1-d array ==> column

array():

========

Syntax:

import numpy

numpy.array(object, dtype = None, copy = True, order = None, subok = False, ndmin = 0)

order:

======

C ==> row major

F ==> Column major

A ==> Any

1-D array:

=========

Syntax:

numpy.array([list/tuple])

2-D array:

==========

Syntax:

numpy.array([[1d],[1d],[1d],...[1d]])

3-D array:

=========

Syntax:

numpy.array([[2d],[2d],[2d],...[2d]])

import numpy as rk

# creating an array with array() by consider the only object parameter

a = rk.array((1,2,3,4))# 1-d array

print(a,type(a))

b = rk.array(10)

print(b,type(b))

# 2D-array

# Collection of more than one 1-d array ==> 2d-array

c = rk.array([[1,2,3],[4,5,6]])

print(c,type(c))

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

print(d,type(d))

# 3-d array

# collection of 2-d arrays

e = rk.array([[[1,2,3,4],[5,6,7,8]],[[9,10,11,12],[13,14,15,16]]])

print(e,type(e))

**Day70: Shape and Strides:**

Array Creation:

===============

array():

=======

Syntax:

numpy.array(object, dtype = None, copy = True, order = None, subok = True/False, ndmin = 0)

order = A (both row and column)

C (in row)

F (in Column)

1d array: [1,2,3]

2d-array: collection of 1-d arrays

[[1,2,3],[3,4,5],[5,6,7]]

3d-array: collection of 2-d arrays

[[[1,2,3],[4,5,6]],[[7,8,9],[10,11,12]]]

Creation of an array by specifying the Dimension value:

=======================================================

Syntax:

numpy.array(object, ndmin = value)

import numpy as rk

# create an array with ndmin value

a = rk.array([1,2,3,4],ndmin = 3)

b = rk.array([[1,2,3,4,5],[5,4,3,2,1]],ndmin = 4)

print(a)

print(b)

===========================================================

shape

=====

==> a built-in parameter in NumPy

which we can use to get the shape of the array

==> shape of array ==> row and column

Syntax:

array-name.shape

Ex:

a =

[[1,2,3],

[4,5,6],

[7,8,9]] ==> 2-d array ==> consisting of three 1d-arrays.

Each 1d-array ==> row

the total rows ==> 3

In this row: three elements

element ==> value at column

total columns => 3

Shape ==> 3 X 3

Ex:

[[[1,2,3],[4,5,6]],

[[3,2,1],[6,5,4]],

[[7,8,9],[10,11,12]]' ==> 3d-array

In this definition:

total 2d-arrays ==> 3 (length of 3d-array)

each 2d-array ==> 2-1darrays (length of 2d-array)

each 1d-array ==> 3-elements (length of 1d-array)

3 X 2 X 3

Ex:

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

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

[[[1,3],[5,7]],[[2,4],[6,8]]]] ==> 4d-array

In this definition:

total number of 3d-arrays ==> 3 (length of 4d-array = 3)

each 3d-array ==> 2 2d-arrays ==> length of 3d-array = 2

each 2d-array ==> 2 1d-arrays ==> length of 2d-array = 2

each 1d-array ==> 2 elements ==> length of 1d-array = 2

Shape ==> 3 X 2 X 2 X 2

import numpy

a = numpy.array([1,2,3,4]) # 1-d array

b = numpy.array([[1,2,3],[4,5,6]]) # 2d-array

c = numpy.array([[[1,2,3,4],[5,6,7,8]],[[4,3,2,1],[8,7,6,5]]]) # 3d-array

d = numpy.array([[[[1,2],[3,4]],[[5,6],[7,8]]],[[[9,10],[11,12]],[[13,14],[15,16]]]]) # 4d-array

print("The Shape of array a = ",a.shape)

print("The Shape of array b = ",b.shape)

print("The Shape of array c = ",c.shape)

print("The Shape of array d = ",d.shape)

============================================================

strides:

========

Syntax:

array-name.strides

Ex:

a =

[[1,2,3],

[4,5,6],

[7,8,9]] ==> 2-d array ==> consisting of three 1d-arrays.

3 X 3

3-rows

and 3-columns

strides ==> 12-bytes, 4-bytes

Ex:

[[[1,2,3],[4,5,6]],

[[3,2,1],[6,5,4]],

[[7,8,9],[10,11,12]]' ==> 3d-array

strides: 24-bytes, 12-bytes, 4-bytes

Ex:

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

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

[[[1,3],[5,7]],[[2,4],[6,8]]]] ==> 4d-array

strides: 32-bytes, 16-bytes, 8-bytes, 4-bytes

import numpy

a = numpy.array([1,2,3,4]) # 1-d array

b = numpy.array([[1,2,3],[4,5,6]]) # 2d-array

c = numpy.array([[[1,2,3,4],[5,6,7,8]],[[4,3,2,1],[8,7,6,5]]]) # 3d-array

d = numpy.array([[[[1,2],[3,4]],[[5,6],[7,8]]],[[[9,10],[11,12]],[[13,14],[15,16]]]]) # 4d-array

print("The Strides of array a = ",a.strides)

print("The Strides of array b = ",b.strides)

print("The Strides of array c = ",c.strides)

print("The Strides of array d = ",d.strides)

**Day71: NumPy Datatypes Part\_01:**

NumPy Datatypes:

=================

1) Boolean datatype ==> bool\_

True/False

1-byte

2) Integer Datatype:

====================

==> Any integer in NumPy can hold either 4 or 8 bytes of memory.

Signed Integer ==> positive and negative values

Unsigned Integer ==> positive values

ex: -10 ==> negative, +10 ==> positive

10 ==> positive

Signed Integer

==============

int\_ ==> 4-bytes/8-bytes

int8 ==> 1-byte ==> 256 values ==> -128 to -1 and 0 to 127 ==> (-2^8)/2 to (2^8)/2 - 1

int16 ==> 2-bytes ==> 2^16 values ==> 65536 values ==> -32768 to -1 & 0 to 32767

==> (-2^16)/2 to (2^16)/2 - 1

int32 ==> 4-bytes ==> (-2^32)/2 to (2^32)/2 - 1

int64 ==> 8-bytes ==> (-2^64)/2 to (2^64)/2 - 1

100 ==> 1byte

Unsigned Integer

================

uint8 ==> 1-byte ==> 0 to 2^8 - 1

uint16 ==> 2-bytes ==> 0 to 2^16 - 1

uint32 ==> 4-bytes ==> 0 to 2^32 - 1

uint64 ==> 8-bytes ==> 0 to 2^64 - 1

Floating-point Datatype:

========================

==> 8-bytes

float\_ ==> 8-bytes

float16 ==> 2-bytes

float32 ==> 4-bytes

float64 ==> 8-bytes

Complex Datatype:

=================

complex\_ ==> 16-bytes

complex64 ==> 8-bytes

complex128 ==> 16-byes

==========================================

Creation of Data object:

========================

dtype():

=======

we can use to create a dtype object

Syntax:

import numpy

numpy.dtype(object, align, copy)

C:

Structures

struct Student{

char name[20]; //20 bytes

int roll;

float marks;

}

==================

# Scalar Type of dtype object

import numpy

dt1 = numpy.dtype(numpy.int16)

dt2 = numpy.dtype(numpy.float32)

dt3 = numpy.dtype(numpy.uint8)

print(dt1)

print(dt2)

print(dt3)

===========================

==> for each built-in datatype:

we have the string notation for identification:

bool\_ ==> 'b'

integer ==> 'i'

unsigned integer ==> 'u'

floating-point ==> 'f'

complex data ==> 'c'

String data ==> 'S'

# Creation of dtype object using string literal

import numpy as np

dt1 = np.dtype(np.uint64)

dt2 = np.dtype('u8')

dt3 = np.dtype(np.float64)

dt4 = np.dtype('f8')

print(dt1)

print(dt2)

print(dt3)

print(dt4)

**Day72: Array Creation with Specified Datatypes:**

dtype()

=======

Creation of Structured Datatype:

================================

Structures:

===========

struct Student{

char name[30];

int roll;

float marks;

}

import numpy

dt = numpy.dtype([('age',numpy.uint8)])

print(dt)

==============================================

Create arrays with dtype object (with specified datatype):

==========================================================

dtype:

=====

==> which we can use this to get the type of an array.

Syntax:

array-name.dtype

import numpy

idt1 = numpy.dtype('i1')

idt2 = numpy.dtype('i2')

idt3 = numpy.dtype('i4')

idt4 = numpy.dtype('i8')

# arrays with signed Integer objects

a1 = numpy.array([1,-2,3,-4],dtype = idt1) # array with signed 8-bit notation

a2 = numpy.array([-1,-2,3,4,-5,6],dtype = idt2)

a3 = numpy.array([0,1,0,2,0,3,0,4],dtype = idt3)

a4 = numpy.array([-1,0,-2,0,-3,0],dtype = idt4)

print("a1 = ",a1)

print("Type = ",a1.dtype)

print("a2 = ",a2)

print("Type = ",a2.dtype)

print("a3 = ",a3)

print("Type = ",a3.dtype)

print("a4 = ",a4)

print("Type = ",a4.dtype)

==================================================

# CREATING THE INTEGER ARRAY FROM other datatype

import numpy

a = numpy.array([1.2,2.1,1.3,3.1,1.4,4.1]) # float array

print("a = ",a)

print("Type = ",a.dtype)

a\_int = numpy.array([1.2,2.1,1.3,3.1,1.4,4.1],dtype = numpy.uint8)

a\_int1 = numpy.array([1.2,2.1,1.3,3.1,1.4,4.1],dtype = 'u4')

print("a\_int = ",a\_int)

print("Type = ",a\_int.dtype)

print("a\_int1 = ",a\_int1)

print("Type = ",a\_int1.dtype)

==========================================================

Type Casting:

=============

==> Also called as "Type Conversion"

which we can use to convert one type of an array to another type of an array.

==> In Two ways:

1) Internal Type Casting/Automatic Type Casting

2) External Type Casting

==> Internal Type Casting can be done by the Python Machine automatically.

a = 112

b = 11.2

c = 0

print(type(a),type(b),type(c))

c = a + b

print(type(a),type(b),type(c))

==> External Type Casting can be performed by the programmer.

In Python:

int(), float(), bool(), complex(), str(), list(), tuple(), set(), dict() etc.

a = 112

b = 11.2

c = 0

print(type(a),type(b),type(c))

c = a + b

print(type(a),type(b),type(c))

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

print(type(a))

a = tuple(a)

print(type(a))

=======================================

In NumPy:

for the type casting:

1) astype()

2) casting functions

ASSIGNMENT:

===========

1) WAP IN PYTHON USING NUMPY TO CREATE AN ARRAYS WITH UNSIGNED INTEGER FORMAT.

2) WAP IN PYTHON USING NUMPY TO CREATE A COMPLEX ARRAY FROM INTEGER ARRAY.

**Day73: Type Casting:**

Type Conversion in NumPy:

========================

two types:

1) Internal Type Conversion/Implicit Type Conversion

=====================================================

import numpy

a = numpy.array([1,2,3,4,5,6],dtype = 'c8')

print("The Type of array = ",a.dtype)

2) External Type Conversion/Explicit Type Conversion

====================================================

1) astype()

2) cast() functions

1) astype()

===========

==> a built-in method in numpy

which we can use for the external type conversion

Syntax:

source\_array\_name.astype(numpy.datatype)

import numpy as np

# creating the array with unsigned integer 32-bit representation

# a = np.array([1,3,5,7,9,11,13,15,17,19,21,23,25],dtype = np.uint32)

a = np.array([1,3,5,7,9,11,13,15,17,19,21,23,25],dtype = 'u4')

print("The Type of the given array definition is = ",a.dtype)

# Type conversion from Unsigned integer to signed integer 64-bits

a\_s\_int = a.astype(np.int64)

print("The Type of the above definition is = ",a\_s\_int.dtype)

# Unsigned Integer to float 32-bit

a\_float = a.astype('f4')

print("The Type of the above definition is = ",a\_float.dtype)

# Signed Integer ==> complex 128-bit

a\_complex = a\_s\_int.astype('c16')

print("The Type of the above definition is = ",a\_complex.dtype)

# Floating ===> Boolean

a\_bool = a\_float.astype('b1')

print("The Type of the above definition is = ",a\_bool.dtype)

===================================================================

Cast functions:

===============

Syntax:

numpy.cast(s-array)

int8() ==> any array to signed 8-bit integer array

int16() ==> any array to signed 16-bit integer array

int32() ==> any array to signed 32-bit integer array

int64() ==> any array to signed 64-bit integer array

uint8() ==> any array to unsigned 8-bit integer array

uint16() ==> to unsigned 16-bit integer array

uint32() ==> to unsigned 32-bit integer array

uint64() ==> to unsigned 64-bit integer array

float32()

float64()

complex64()

complex128()

import numpy as np

a = np.array([1,3,5,7,9,11,13,15])

print("The Type of defined array = ",a.dtype)

# integer 32-bit array ==> unsigned integer 64-bit

a\_u = np.uint64(a)

print("Type = ",a\_u.dtype)

# signed integer ==> complex64

a\_c = np.complex64(a)

print("Type = ",a\_c.dtype)

# unsigned integer ==> float

a\_u\_f = np.float16(a\_u)

print("Type = ",a\_u\_f.dtype)

a\_bool = np.bool\_(a)

print("Type = ",a\_bool.dtype)

**DAY74: Array Creation Routines:**

Array Creation Routines:

========================

1) zeros()

==========

==> built-in function

which we can use to create an array with only zeros.

Syntax:

numpy.zeros(shape, dtype = float)

import numpy

a = numpy.zeros(6)

print(a)

print(a.dtype)

b = numpy.zeros(10,dtype = 'i4')

print(b)

print(b.dtype)

c = numpy.zeros(5,dtype = 'c8',order = 'F')

print(c)

print(c.dtype)

d = numpy.zeros((3,4),dtype = 'i4', order = 'C')

print(d)

e = numpy.zeros((3,4,5),dtype = 'i2')

print(e)

========================================

ones()

======

==> built-in function

creating the array with only '0nes'

Syntax:

numpy.ones(shape, dtype, order)

import numpy

a = numpy.ones(6)

print(a)

print(a.dtype)

b = numpy.ones(10,dtype = 'c8')

print(b)

print(b.dtype)

c = numpy.ones((3,3),dtype = 'i4')

print(c)

print(c.dtype)

d = numpy.ones([2,3,4],dtype = numpy.int8)

print(d)

print(d.dtype)

===============================================

In Python:

range() ==> can generate the range of values

ex: range(10) ==> generate values from 0 to 9

range(1,20) ==> generate values from 1 to 19

range(1,100,10) ==> generate values from 1 to 100 with the difference of 10.

arrange():

=========

==> built-in function

which we can use to create an array with range of values.

Syntax:

numpy.arange(start, stop, step, dtype)

import numpy

a = numpy.arange(10)

print(a)

print(a.dtype)

b = numpy.arange(6,dtype = 'f8')

print(b)

print(b.dtype)

c = numpy.arange(1,20)

print(c)

print(c.dtype)

d = numpy.arange(1,100,10)

print(d)

# e = numpy.arange([(10,),(1,)])

# print(e)

=======================================

linspace()

==========

==> built-in function

we can use to create an array with range values.

Syntax:

numpy.linspace(start, stop, num, endpoint = True)

import numpy

a = numpy.linspace(1,10, num = 6) # endpoint ==> optional

# endpoint = True

# endpoint = True ==> linspace() can consider the stop value also in array creation

print(a)

print(a.dtype)

b = numpy.linspace(1,10,num = 7, endpoint = False,dtype = 'i4')

# endpoint = False ==> no need to consider the stop value in array creation

print(b)

print(b.dtype)

=========================================================

In Python:

Random Module

random() ==> generate random floating-point values from 0 to 1

dice ==> when we can roll a dice:

generate: 1 | 2 | 3 | 4 | 5 | 6

rand():

======

==> a built-in function in NumPy with "random" module

use to create an array with random float numbers (0 to 1).

Syntax:

numpy.random.rand(d0, d1, d2, d3, .....)

import numpy

a = numpy.random.rand() # can generate a value (float) from 0 to 1

print(a)

# one dimensional array

b = numpy.random.rand(6)

print(b)

print(b.dtype)

# two dimensional array

c = numpy.random.rand(3,3)

print(c)

# four dimensional array

d = numpy.random.rand(1,3,4)

print(d)

=================================================

empty()

=======

==> built-in function

can use to create an array with random values of any specified datatype

Syntax:

numpy.empty(shape, dtype)

import numpy

a = numpy.empty(10)

print(a)

b = numpy.empty((3,4))

print(b)

c = numpy.empty([3,3,3],dtype = 'i1')

print(c)

print(c.dtype)

**DAY75: Array Creation from Existing Data:**

Creation of Array from Existing Data:

=====================================

From List:

==========

# WAP TO CREATE THE ARRAY FROM LIST DATA

import numpy

ld1 = [1,2,3,4,5,6,7,8,9,10]

print(type(ld1))

# array from list

a1 = numpy.array(ld1)

print("The Array = ",a1)

ld2 = [[3,2,1],[6,5,4],[9,8,7]] # Nested list

print(type(ld2))

a2 = numpy.array(ld2)

print("The Array = ",a2)

ld3 = [12,True,1.23,12-23j,'python'] # list with heterogeneous elements

a3 = numpy.array(ld3)

print("The Array = ",a3)

ld4 = [[True, 10, 1.12],[1-2j,12,1.23],['a',112,False]] # nested list with heterogeneous elements

a4 = numpy.array(ld4)

print("The Array = ",a4)

from Tuple:

===========

import numpy

td = (1,11,111,1111,11111,1111,111,11,1)

sd = {1,3,5,7,9,9,7,5,3,1,1,2,3,4,5}

dd = {'a':111,'b':222,'c':333,'d':444,'e':555}

print(type(td))

print(type(sd))

print(type(dd))

# array from the tuple

a1 = numpy.array(td)

a2 = numpy.array(sd)

a3 = numpy.array(dd)

print("The Array = ",a1)

print("The Array = ",a2)

print("The Array = ",a3)

=====================================================

asarray():

==========

use to convert various python objects like: list, tuple, array etc. into array.

Syntax:

numpy.asarray(object, dtype)

import numpy

ld = [1,11,111,1111,11111,1111,111,11,1]

ld1 = [111,True,1.12,False,12-23j,'python']

a1 = numpy.asarray(ld)

a2 = numpy.asarray(ld1)

print("The Array = ",a1)

print("The Array = ",a2)

print("The Type = ",a2.dtype)

print("The Type = ",a1.dtype)

frombuffer():

=============

==> a built-in function

can use to create an array from bytes object or bytes array.

bytes object ==>

bytes() ==> bytes object

Syntax:

numpy.frombuffer(buffer\_object, dtype)

import numpy

b1 = bytes([1,2,3,4,5]) # bytes object

# b2 = bytes("Python")

b2 = b'Python'

a1 = numpy.frombuffer(b1,dtype = 'S1')

a2 = numpy.frombuffer(b2,dtype = "S1")

print("a1 = ",a1)

print("a2 = ",a2)

==============================================

fromiter():

===========

==> built-in function

can use to create an array from iterable object.

Syntax:

numpy.fromiter(iterable-object, dtype)

import numpy

def my\_generator(n):

for i in range(n):

yield i

a = numpy.fromiter(my\_generator(7),dtype = 'i1')

print("The Array = ",a)

print("The Type = ",a.dtype)

================================================

copy():

======

==> built-in function

can use to create a new array from another array.

Syntax:

numpy.copy(s-array)

import numpy

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

a\_copy = numpy.copy(a)

print("The Arrays are = ")

print(a)

print(id(a))

print(a\_copy)

print(id(a\_copy))

**DAY76: Array\_Creation\_Looping\_Arrays:**

view():

=======

==> built-in function

can use to create an array from another array

the array object in view is same as original array but possible to change its type.

a = [1,3,5,7,9]

a\_copy = [1,3,5,7,9]

a\_view = [1.0,3.0,5.0,7.0,9.0]

Syntax:

source-array.view(dtype)

import numpy

a = numpy.array([1.2,2.1,1.3,3.1,1.4,4.1])

print("The Original Array = ",a)

print("Type = ",a.dtype)

# b = a.copy(dtype = 'i2')

b = a.copy()

print("b = ",b)

print("type = ",b.dtype)

c = a.view(dtype = 'i1')

print("c = ",c)

print("Type = ",c.dtype)

==========================================

reshape():

==========

array ==> 3 X 4 ==> 12

reshape to ==> 4 X 3 ==> 12

reshape ==> 6 X 2 ==> 12

Note:

====

the reshape parameters product must be equal to total number of elements in the original array,

then only the reshape is possible.

==> reshape() built-in function

can be use to reshape the original array

and create the new array.

Syntax:

original\_array.reshape((parameters))

for 2d-array: x, y

for 3d-array: x, y, z

import numpy

a = numpy.array([1,2,3,4,5,6]) # 1d-array

b = numpy.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]]) # 2d-array

print("a = ",a)

print("b = ",b)

r1 = a.reshape((3,2))

r2 = a.reshape((2,3))

# r3 = a.reshape((3,3))

r3 = b.reshape((12,))

r4 = b.reshape((1,3,4))

print("r1 = ",r1)

print("r2 = ",r2)

print("r3 = ",r3)

print("r4 = ",r4)

================================================

Array Indexing:

===============

import numpy

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

b = numpy.array([[1,2,3],[4,5,6],[7,8,9]]) # 2d-array

# Positive Indexing values

print(a[0],a[1],a[2],a[3],a[4])

print(b[0],b[1],b[2])

print(b[0][0],b[0][1],b[0][2])

print(b[1][0],b[1][1],b[1][2])

print(b[2][0],b[2][1],b[2][2])

# Negative Indexing Values

print(a[-1],a[-2],a[-3],a[-4],a[-5])

print(b[-1],b[-2],b[-3])

print(b[-1][0],b[-1][1],b[-1][2])

=============================================

Array Slicing:

==============

import numpy

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

print(a[::1])

print(a[::-1])

print(a[2:6])

print(a[-10:-4])

=================================================

# WAP TO CREATE THE ARRAY WITH RANGE OF OBJECTS.

import numpy

objects = range(1,11)

a = numpy.array(objects)

b = numpy.array(range(10,101,10))

print("Array with range of objects = ",a)

print("Array with range of objects = ",b)

====================================================

logspace():

===========

log10

log2

==> built-in function

can use to generate an array with log values

these objects can be evenly spaced.

Syntax:

numpy.logspace(start, stop, num, endpoint = True, base = 10/2, dtype)

import numpy

a = numpy.logspace(1,11,6,base = 2)

print("a = ",a)

===================================================

Array-Iteration:

================

1) Using for loop:

==================

import numpy

a = numpy.array([1,3,5,7,9]) # 1d-array

b = numpy.array([[1,2,3],[4,5,6],[7,8,9]])

c = numpy.array([[[1,2,3,4],[5,6,7,8]],[[9,10,11,12],[13,14,15,16]]])

for i in a:

print(i)

for i in a[::-1]:

print(i)

for i in b:

print(i)

for i in b:

for j in i:

print(j,end = "\t")

print()

print("Printing 2d-arrays:")

for i in c:

print(i)

print("Printing 1D-array:")

for i in c:

for j in i:

print(j)

print("3d-Array Elements = ")

for i in c:

for j in i:

for k in j:

print(k,end = "\t")

print()

**DAY77: Array Manipulations Part\_01:**

Assignment:

===========

WAP TO CREATE AN 1-D ARRAY AND PRINTS ITS ELMENTS ALONG WITH BOTH POSITIVE AND NEGATIVE INDEX USING WHILE LOOP.

Hint:

=====

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

Output:

The Element at a positive index '0' and at negative index '-5' = 1

======================================================

nditer():

=========

1d-array:

for i in a:

print(i)

2d-array:

for i in a:

for j in i:

print(j)

3d-array:

for i in a:

for j in i:

for k in j:

print(k)

nditer(): a built-in function

can used generate an iterative object

to iterate on individual elements of any dimensional array.

Syntax:

numpy.nditer(array-name)

import numpy

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

print("Printing Elements of 1d-array:")

for i in numpy.nditer(a):

print(i)

=========================

import numpy

a = [[1,2,3],[4,5,6],[7,8,9]]

for i in numpy.nditer(a):

print(i)

==================================

Array Concatenation:

====================

concatenate()

Syntax:

numpy.concatenate((a1,a2,a3,...), axis = 0/1/2,...)

2d-array:

0

1

ex:

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

b = [[4,5,6][7,8,9]]

a concate b ==> axis = 0

[[[1,2,3]][4,5,6],[[4,5,6][7,8,9]]]

axis=1[[[1,2,3],[4,5,6]],[[4,5,6],[7,8,9]]]

3d-array:

0

1

2

import numpy

a1 = numpy.array([1,2,3])

a2 = numpy.array([4,5,6])

r1 = numpy.concatenate((a1,a2))

r2 = numpy.concatenate((a1,a2),axis = 0)

# r3 = numpy.concatenate((a1,a2),axis = 1)

print("r1 = ",r1)

print("r2 = ",r2)

# print("r3 = ",r3)

========================

import numpy

a1 = numpy.array([[0,2],[4,6],[8,10]])

a2 = numpy.array([[1,3],[5,7],[9,11]])

r1 = numpy.concatenate((a1,a2))

r2 = numpy.concatenate((a1,a2),axis = 0)

r3 = numpy.concatenate((a1,a2),axis = 1)

# r4 = numpy.concatenate((a1,a2),axis = 2)

print("r1 = ",r1)

print("r2 = ",r2)

print("r3 = ",r3)

# print("r4 = ",r4)

===============================================

Array Stacking:

===============

==> combining multiple array into one by making the higher-dimension.

stack():

=======

Syntax:

numpy.stack((a1,a2,a3,...),axis = 0/1/2,..)

import numpy

a1 = numpy.array([1,2,3])

a2 = numpy.array([4,5,6])

r1 = numpy.stack((a1,a2))

r2 = numpy.stack((a1,a2),axis = 1)

print("r1 = ",r1)

print("r2 = ",r2)

=======================

import numpy

a1 = numpy.array([[1,2,3,4],[5,6,7,8]])

a2 = numpy.array([[9,10,11,12],[13,14,15,16]])

r1 = numpy.stack((a1,a2))

r2 = numpy.stack((a1,a2),axis = 1)

r3 = numpy.stack((a1,a2),axis = 2)

print("r1 = ",r1)

print("r2 = ",r2)

print("r3 = ",r3)

======================================

Array Splitting:

================

split():

=======

[[1,2,3],[4,5,6],[7,8,9]]

[[1,2],[3,4],[5,6],[7,8],[9,10]]

Syntax:

numpy.split(array\_name, indices, axis)

============================================

import numpy

a = numpy.arange(9).reshape(3,3)

b = numpy.array([[1,2],[3,4],[5,6],[7,8],[9,10]])

print("original Array = ",a)

print("Original Array = ",b)

a\_split = numpy.split(a,3)

b\_split = numpy.split(b,5)

print("splitted array = ",a\_split)

print("splitted array = ",b\_split)

**DAY78: Array Manipulations Part\_02:**

Broadcasting:

=============

refers to the ability of performing operations on arrays with different shapes

Ex:

+

1-d + 2-d

==> while the operation, the lower dimension array can automatically expand according to the higher dimension.

Rules for Broadcasting:

======================

1) If arrays are with different dimensions:

the shape of smaller dimension is must be padded with '1' on the left side until this match with higher dimension array.

[[1 1 1 1],[1 1 1 1],[1 2 3 4]] + [[1 2 3 4],[5 6 7 8],[9 10 11 12]]

2) The size of each dimension must either be the same or one of the dimension must be same.

3) Broadcasting can be applied from last dimension to first dimension.

# WAP IN NUMPY TO ADD A SCALAR TO AN ARRAY.

"""

[1 2 3] + 10

[1,2,3] + [1,1,10]

"""

import numpy

a = numpy.array([1,2,3])

b = numpy.array([[1,2,3],[4,5,6],[7,8,9]])

result = a + 10

res1 = b + 100

print(result)

print(res1)

==============================

import numpy as np

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

b = np.array([10,20,30,40]) # [[1,1,1,1],[1,1,1,1],[10,20,30,40]]

result = a + b

print("The Sum of two differently shaped arrays = ",result)

==============================

# Broadcasting with multi-dimensional arrays

import numpy

a = numpy.ones((2,3,4))

b = numpy.arange(4)

print("The Given array = ",a)

print("The Given array = ",b)

result = a + b

print(result)

=================================================

Arithmetic Operations:

======================

Array Addition:

===============

import numpy

a = numpy.array([10,20,30,40,50])

b = numpy.array([50,40,30,20,10])

sum = a + b

print("The Sum = ",sum)

===============================

import numpy as np

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

b = np.array([[10,9,8,7,6],[5,4,3,2,1]])

sum = a+b

print("The Sum = ",sum)

=============================

Array Subtraction:

==================

import numpy as np

a = np.arange(1,25).reshape(2,3,4)

b = np.arange(24).reshape(2,3,4)

print("a = ",a)

print("b = ",b)

difference = a-b

print(difference)

==============================

Array Multiplication:

=====================

import numpy

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

b = numpy.array([5,6,7,8])

product = a \* b

print("The Product = ",product)

======================================

Array Division:

===============

import numpy as np

a = np.arange(5)

b = np.arange(1,6)

print("a = ",a)

print("b = ",b)

print(a/b)

print(a//b)

print(a%b)

============================

Array Power Operation:

============================

Syntax:

a \*\* b

import numpy

a = numpy.arange(5)

b = numpy.arange(1,6)

print("a = ",a)

print("b = ",b)

print(a \*\* b)

================================================

# Sorting

import numpy as np

a = np.array([[100,12,301],[99,88,101]])

print("The Array before the sorting is = ",a)

# sort()

s\_a1 = np.sort(a) # forward sorting

print("Original Array = ",a)

print("The Sorted Array = ",s\_a1)

print(np.sort(a,axis = 1))

=============================================================

**DAY79: Date Time Handling:**

Date & Time:

============

==> To handle the date and time:

there are three modules in Python Library:

1) time module

2) calendar module

3) datetime module

1) time module:

===============

time():

=====

return the total time in float.

==> consider the time period from Jan-01, 1970 12 am to till

Syntax:

import time

time.time()

import time

t = time.time()

localtime():

===========

which includes:

year, month, date, time (hours, minutes, seconds..)

Syntax:

time.localtime()

return the local time in tuple format.

import time

lt = time.localtime()

print(lt)

asctime():

==========

==> use to print the time in a format.

Syntax:

time.asctime(local-time)

import time

lt = time.localtime()

print("Local time in Tuple Format : ",lt)

t = time.asctime(lt)

print("The Local Time in time format : ",t)

2) calendar Module:

===================

month():

=======

required to display the calendar of the specified year and month

Syntax:

import calendar

calendar.month(year, month-number)

month-number ==> 1 to 12

import calendar

month = calendar.month(2024,11)

print(month)

print(calendar.month(2000,6))

===============================================

3) datetime Module:

===================

date():

=======

Syntax:

datetime.date(year, month, day)

month ==> 1-12

day ==> 1-30/1-31/1-28

import datetime

d1 = datetime.date(1993, 6, 20)

d2 = datetime.date(2050, 11,4)

print(d1)

print(d2)

now():

=====

Syntax:

datetime.datetime.now()

import datetime

n = datetime.datetime.now()

print(n)

===========================================

Minimum date and Maximum date:

==============================

from datetime import date

min\_date = date.min

max\_date = date.max

print(min\_date)

print(max\_date)

=============================================

Individual Properties to handle date and time:

=============================================

import datetime

t = datetime.datetime.now()

print(t)

print("Year = ",t.year)

print("Month = ",t.month)

print("Day = ",t.day)

print("Hours = ",t.hour)

print("Minutes = ",t.minute)

print("Seconds = ",t.second)

============================================

today():

=======

Syntax:

date.today()

from datetime import date

t = date.today()

tf1 = date.fromisoformat('1993-06-20')

tf2 = date.fromisoformat('20051119')

tf3 = date.fromisoformat('2005-W16-4')

print(t)

print(tf1)

print(tf2)

print(tf3)

===================================================

Format Specifiers:

==================

%a ==> day name in short form

%A ==> day name in full form

%w ==> week day number

0 to 6

sun = 0 mon = 1 tue = 2 wed = 3 thu = 4 fri = 5 sat = 6

%d ==> month day number

30 days ==> 1 to 30

31-days ==> 1 to 31

28-days ==> 1 to 28

%b ==> month name in short form

%B ==> month name in full form

%m ==> month number

1 to 12

%y ==> Year in short form

%Y ==> Year in full form

%H ==> Time in 24 hours format (0 to 23)

%I ==> Time in 12 hour format (0 to 11)

%p ==> am/pm

%M ==> minute

%S ==> Seconds

%Z

import datetime

lt = datetime.datetime.now()

print(lt.strftime("%a"))

print(lt.strftime("%A"))

print(lt.strftime("%w"))

print(lt.strftime("%d"))

print(lt.strftime("%b"))

print(lt.strftime("%B"))

print(lt.strftime("%m"))

print(lt.strftime("%y"))

print(lt.strftime("%Y"))

print(lt.strftime("%H"))

print(lt.strftime("%I"))

print(lt.strftime("%p"))

print(lt.strftime("%M"))

print(lt.strftime("%S"))

print(lt.strftime("%Z"))

print("The Time in the format:",lt.strftime("%B-%d-%Y %H:%M:%S %p"))

**Day80: Iterator:**

ld = [1,3,5,7,9]

for l in ld:

print(l)

# iterable can make the traversing on the collection like list, tuple, set etc, element by element

# Iterable can return only one element/value at a time.

print()

i = 0

while i < len(ld):

print(ld[i])

i+=1

ITERATOR:

=========

==> Iterator is an object

which can enable the traversing on the collection like: list, tuple, dictionary etc.

==> Iterator Object, can return only one element at a item.

==> can also return the stream of data.

==> two methods:

iter() ==> can return the iterator object to iterate over the data

next() ==> can return the next element in the collection.

d1 = iter("Hello")

d2 = iter([1,3,5,7,9])

print(d1)

print(d2)

print(d1.\_\_next\_\_())

print(d1.\_\_next\_\_())

print(d1.\_\_next\_\_())

print(d1.\_\_next\_\_())

print(d1.\_\_next\_\_())

======================================

Iterable Vs Iterator:

=====================

Iterable can return only one element at a time.

Iterable cannot use the memory efficiently.

Iterator can return the stream of data at a time.

Iterator can use the memory efficiently.

data = iter(range(5))

print(data)

print(data.\_\_next\_\_())

print(data.\_\_next\_\_())

print(data.\_\_next\_\_())

print(data.\_\_next\_\_())

print(data.\_\_next\_\_())

print(data.\_\_next\_\_())

===================================================

# d = 100

# for i in d:

# print(i)

# Iterable like: while, for cannot iterate on the integer (primitive data)object.

d = iter(100)

print(d)

# Like iterable, iterator can also not iterate on the primitive data object.

=================================================

StopIteration:

==============

==> an exception

can be occurred when the number of iterations exceeded over the collection.

Handling the "StopIteration" Exception:

========================================

d = iter((1,3,5,7,9))

print(d)

while True:

try:

n = next(d)

print(n)

except StopIteration:

break

=======================================

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

print(i) # 1 3 5 7 9

# WAP TO GENERATE ODD NUMBERS FROM THE GIVEN RANGE USING ITERATOR.

class OddNumberGeneration:

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

self.start = -1

self.last = last

def \_\_iter\_\_(self):

return self

def \_\_next\_\_(self):

if self.start < self.last - 1:

self.start += 2 # self.start = self.start + 2 # -1 + 2 = 1 3 5 7

return self.start

else:

raise StopIteration

obj = OddNumberGeneration(10)

while True:

try:

x = next(obj)

print(x)

except StopIteration:

break

**DAY81: Closures**:

Closures:

=========

what is closure?

================

==> a nested function

Syntax:

def outer\_function():

implementation

def inner\_function():

implementation

==> has access a variable from an enclosing function that has finished its execution.

==> That variable not bound in the local scope.

==> If the variable is immutable type:

we can use "nonlocal" keyword to modify that immutable variable.

Advantage:

==========

1) can avoid the usage of global variable

2) provide some form of the data hiding.

Nested Function:

================

def fun1(): # Outer function

print("This is the Outer function")

def fun2(): # Inner Function

print("This is the Inner function")

print("Hi")

print("Good Morning")

print("This is Python Advanced Class")

fun2()

fun1()

==========================================

Variable Scope:

===============

def outerFunction(x): # x = 1122

y = 1221

def innerFunction(z): # z = 1234

return x + y + z

return innerFunction

returningFunction = outerFunction(1122) # function aliasing returningFunction = innerFunction

result = returningFunction(1234)

print(result)

=================================

def outer(name):

# name = local variable to the "outer" function

name = "Ashok IT"

def inner():

print("Name = ",name)

return inner

res1 = outer("Python") # function aliasing

res1()

=======================================================

Non-local:

=========

def f1():

a = 112 # Immutable

def f2():

nonlocal a

b = 121

a = a + b

return a

return f2

r1 = f1()

print(r1())

print(r1())

print(r1())

print(r1())

======================================================================

**Day82: Regular Expressions:**

Regular Expressions (RegEx):

============================

==> denoted as "RegEx".

==> is a sequence of characters that uses a search pattern to find a string or set of string.

==> RegEx ==> built-in module named as "re"

module ==> a collection of files, classes, functions etc.

==> while using the RegEx, we must import this module.

Syntax:

import re

import re

s = "Python Fullstack at Ashok IT"

m = re.search(r'at',s)

m1 = re.search('Fullstack',s)

print("The Start Index = ",m.start()) # 17

print("The End Index = ",m.end()) # 18

# range(start,end) ==> range() start to generate value from given "start" value and end with "end" value - 1

print("The Start Index = ",m1.start()) # 17

print("The End Index = ",m1.end()) # 18

===============================================

Meta Characters:

================

\

===

import re

s = "Ashok.IT"

m1 = re.search(r'.',s)

print(m1)

m1 = re.search(r'\.',s)

print(m1)

=================================

[]

==

ex: [abc]==> set of characters

[0,7] ==> range

[a-d] ==> range of alphabets ==> [abcd]

import re

s = "The Cat Sat on a wall."

p = "[b-r]" # bcdefghijklmnopqr

p1 = "[bde]"

result = re.findall(p,s)

r1 = re.findall(p1,s)

print(result)

print(r1)

===============================

^

===

import re

p = r'^The'

s = ['The quich brown box','The lazy dog','A quick brown fox']

for i in s:

if re.match(p,i):

print(i)

else:

print("Not found")

=================================

.

===

a.b

"python"

s = "p.t" ==> pyt

import re

s = "The quick brown fox jumps over the lazy dog"

p = r"brown.fox"

p1 = r"b.o"

m = re.search(p,s)

if m:

print("Match Found")

else:

print("No Match found.")

print(m)

m1 = re.search(p1,s)

print(m1)

====================

?

===

ex: ab?c

import re

s = "The quick brown fox jumps over the lazy dog"

p = r"b?o"

m = re.search(p,s)

print(m)

**Day83: Tkinter Part\_01:**

Tkinter:

========

GUI ==> Graphical User Interface

How to use the Tkinter:

=======================

1) Import the tkinter module

=============================

Python-2.x ==> Tkinter

Python-3.x ==> tkinter

Syntax:

import tkinter

or

from tkinter import \*

2) we need to create the main object/window (container)

3) we can add widgets to the main object/window.

4) Apply event trigger to the widget

===========================================

First Tkinter GUI Application:

==============================

Tk():

=====

==> used to create the main window

Syntax:

object-name/window-name = Tk(screenname = None, baseName = None, className = "Tk", useTk = 1)

mainloop():

===========

window ==> widget ==> to run the GUI app

mainloop() ==> Infinite loop

import tkinter

obj = tkinter.Tk() # can add new window

obj.mainloop()

===============================================

Adding of Widgets:

==================

1) Label():

===========

==> can used to display the box where we can put text/image.

text/image which can be updated any time as per the code.

Syntax:

object = Label(master, option = value)

from tkinter import \*

# main window object

obj = Tk()

# adding the widget

widget = Label(obj, text = "Welcome To Ashok IT")

widget.pack()

obj.mainloop()

============================================

Button():

=========

Syntax:

object = Button(master, option = value)

import tkinter as tk

obj = tk.Tk()

obj.title("Counting Seconds")

btn = tk.Button(obj, text = "File", width = 10, command = obj.destroy)

btn.pack()

obj.mainloop()

==================================

Entry():

========

==> used to enter the single line text from the user.

Syntax:

object = Entry(master, option = value)

from tkinter import \*

master = Tk()

Label(master, text = "First Name").grid(row = 0)

Label(master, text = "Last Name").grid(row = 1)

e1 = Entry(master)

e2 = Entry(master)

e1.grid(row = 0, column = 1)

e2.grid(row = 1, column = 1)

mainloop()

===========================================

Radiobutton():

=============

Gender: Male Female Other

Syntax:

object = Radiobutton(master, option = value)

from tkinter import \*

obj = Tk()

v = IntVar()

Radiobutton(obj,text="Male",variable=v,value=1).pack()

Radiobutton(obj, text = "Female", variable = v, value = 2).pack()

Radiobutton(obj,text = "Others", variable = v, value = 3).pack()

mainloop()

==============================

Checkbutton():

==============

Syntax:

object = Checkbutton(master, option = value)

from tkinter import \*

master = Tk()

v1 = IntVar()

Checkbutton(master, text = "Reading", variable = v1).grid(row = 0, sticky = W)

v2 = IntVar()

Checkbutton(master, text = "Surfing", variable = v2).grid(row = 1, sticky = W)

mainloop()

===============================

Listbox():

=========

Syntax:

object = Listbox(master, option = value)

from tkinter import \*

top = Tk()

Lb = Listbox(top)

Lb.insert(1,"Python")

Lb.insert(2,"Java")

Lb.insert(3,"C++")

Lb.insert(4,"C#")

Lb.insert(5, "Any Other")

Lb.pack()

top.mainloop()

**Day84:** **Tkinter Part\_02:**

Tkinter:

========

Adding of Widgets

=================

Scrollbar:

==========

Syntax:

object = Scrollbar(master, option = value)

from tkinter import \*

window = Tk()

scroll = Scrollbar(window)

scroll.pack(side = RIGHT, fill = Y)

myList = Listbox(window, yscrollcommand = scroll.set)

for i in range(100):

myList.insert(END,'this is line number'+str(i))

myList.pack(side = LEFT,fill = BOTH)

scroll.config(command = myList.yview)

mainloop()

==========================

Menu():

=======

Syntax:

object = Menu(master, option = value)

from tkinter import \*

window = Tk()

m = Menu(window)

window.config(menu = m)

fileMenu = Menu(m)

m.add\_cascade(label = "File",menu = fileMenu)

fileMenu.add\_command(label = "New")

fileMenu.add\_command(label = "Open")

fileMenu.add\_command(label = "Save")

fileMenu.add\_command(label = "Save as")

mainloop()

================================

Combobox()

==========

from tkinter import \*

from tkinter import ttk

def on\_select(event):

selected\_item = combo\_box.get()

label.config(text = "Selected Item: "+selected\_item)

window = Tk()

window.title("Combobox Application")

label = Label(window, text = "Gender")

label.pack(pady = 10)

cb = ttk.Combobox(window, values = ["Male","Female", "Other"])

cb.pack(pady = 5)

cb.set("Default")

cb.bind("<<Combobox selected>>",on\_select)

mainloop()

===================================

Message():

==========

from tkinter import \*

window = Tk()

msg = "Hello, Welcome To Ashok IT"

m = Message(window, text = msg)

m.config(bg = "lightgreen")

m.pack()

mainloop()

===================================

Color Option in Tkinter:

========================

from tkinter import \*

window = Tk()

window.title("Color Option using Tkinter")

btn = Button(window,text = "Click Here",activebackground = "blue", activeforeground = "white")

btn.pack()

label = Label(window, text = "Hello", bg = "lightblue", fg = "red")

label.pack()

entry = Entry(window, selectbackground = "lightblue",selectforeground = "black")

entry.pack()

mainloop()

**Day85: Simple Calculator with Tkinter:**

def addition(a,b):

return a+b

addition(12,13)

from tkinter import \*

import math

class Calculator:

def getandreplace(self):

self.expression = self.e.get()

self.newtext = self.expression.replace('/','/')

self.newtext = self.newtext.replace('x','\*')

def equals(self):

self.getandreplace()

try:

self.value = eval(self.newtext)

except SyntaxError or NameError:

self.e.delete(0,END)

self.e.insert(0,'Invalid Input!')

else:

self.e.delete(0,END)

self.e.insert(0,self.value)

def squareroot(self):

self.getandreplace()

try:

self.value = eval(self.newtext)

except SyntaxError or NameError:

self.e.delete(0,END)

self.e.insert(0,'Invalid Input!')

else:

self.sqrtval = math.sqrt(self.value)

self.e.delete(0,END)

self.e.insert(0,self.sqrtval)

def square(self):

self.getandreplace()

try:

self.value = eval(self.newtext)

except SyntaxError or NameError:

self.e.delete(0,END)

self.e.insert(0,'Invalid Input')

else:

self.sqval = math.pow(self.value,2)

self.e.delete(0,END)

self.e.insert(0,self.sqval)

def clearall(self):

self.e.delete(0,END)

def clear1(self):

self.txt = self.e.get()[::-1]

self.e.delete(0,END)

self.e.insert(0,self.txt)

def action(self,argi):

self.e.insert(END,argi)

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

master.title('Calculator')

master.geometry()

self.e = Entry(master)

self.e.grid(row = 0,column = 0, columnspan = 6, pady = 3)

self.e.focus\_set()

Button(master,text = "=",width = 11, height = 3,fg = "blue",bg = "orange",command = lambda:self.equals()).grid(row = 4,column = 4,columnspan = 2)

Button(master,text = "AC", width = 5, height = 3, fg = "red",bg = "light green", command = lambda:slef.clearall()).grid(row = 1, column = 4)

Button(master,text = "C", width = 5,height = 3, fg = "red", bg = "light green",command = lambda:self.clear1()).grid(row = 1, column = 5)

Button(master,text = "+",width = 5,height = 3,fg = "blue",bg = "orange",command = lambda:self.action('+')).grid(row = 4, column = 3)

Button(master,text = "X",width = 5,height = 3,fg = "blue",bg = "orange",command = lambda:self.action('X')).grid(row = 2, column = 3)

Button(master,text = "-",width = 5,height = 3,fg = "red",bg = "light green",command = lambda:self.action('-')).grid(row = 3, column = 3)

Button(master,text = "Ã·",width = 5,height = 3,fg = "blue",bg = "orange",command = lambda:self.action('/')).grid(row = 1, column = 3)

Button(master,text = "%",width = 5,height = 3,fg = "red",bg = "light green",command = lambda:self.action('%')).grid(row = 4, column = 2)

Button(master,text = "7",width = 5,height = 3,fg = "blue",bg = "orange",command = lambda:self.action('7')).grid(row = 1, column = 0)

Button(master,text = "8",width = 5,height = 3,fg = "red",bg = "light green",command = lambda:self.action('8')).grid(row = 1, column = 1)

Button(master,text = "9",width = 5,height = 3,fg = "blue",bg = "orange",command = lambda:self.action('9')).grid(row = 1, column = 2)

Button(master,text = "4",width = 5,height = 3,fg = "red",bg = "light green",command = lambda:self.action('4')).grid(row = 2, column = 0)

Button(master,text = "5",width = 5,height = 3,fg = "blue",bg = "orange",command = lambda:self.action('5')).grid(row = 2, column = 1)

Button(master,text = "6",width = 5,height = 3,fg = "red",bg = "light green",command = lambda:self.action('6')).grid(row = 2, column = 2)

Button(master,text = "1",width = 5,height = 3,fg = "red",bg = "light green",command = lambda:self.action('1')).grid(row = 3, column = 0)

Button(master,text = "2",width = 5,height = 3,fg = "red",bg = "light green",command = lambda:self.action('2')).grid(row = 3, column = 1)

Button(master,text = "3",width = 5,height = 3,fg = "red",bg = "light green",command = lambda:self.action('3')).grid(row = 3, column = 2)

Button(master,text = "0",width = 5,height = 3,fg = "red",bg = "light green",command = lambda:self.action('0')).grid(row = 4, column = 0)

Button(master,text = ".",width = 5,height = 3,fg = "red",bg = "light green",command = lambda:self.action('.')).grid(row = 4, column = 1)

Button(master,text = "(",width = 5,height = 3,fg = "red",bg = "light green",command = lambda:self.action('(')).grid(row = 2, column = 4)

Button(master,text = ")",width = 5,height = 3,fg = "red",bg = "light green",command = lambda:self.action(')')).grid(row = 2, column = 5)

Button(master,text = "?",width = 5,height = 3,fg = "red",bg = "light green",command = lambda:self.action('?')).grid(row = 3, column = 4)

Button(master,text = ":",width = 5,height = 3,fg = "red",bg = "light green",command = lambda:self.action(':')).grid(row = 3, column = 5)

window = Tk()

obj = Calculator(window)

window.mainloop()

**Day86: Random Password Generator with Tkinter:**

Pyperclip Module

================

==> cross-platform python module

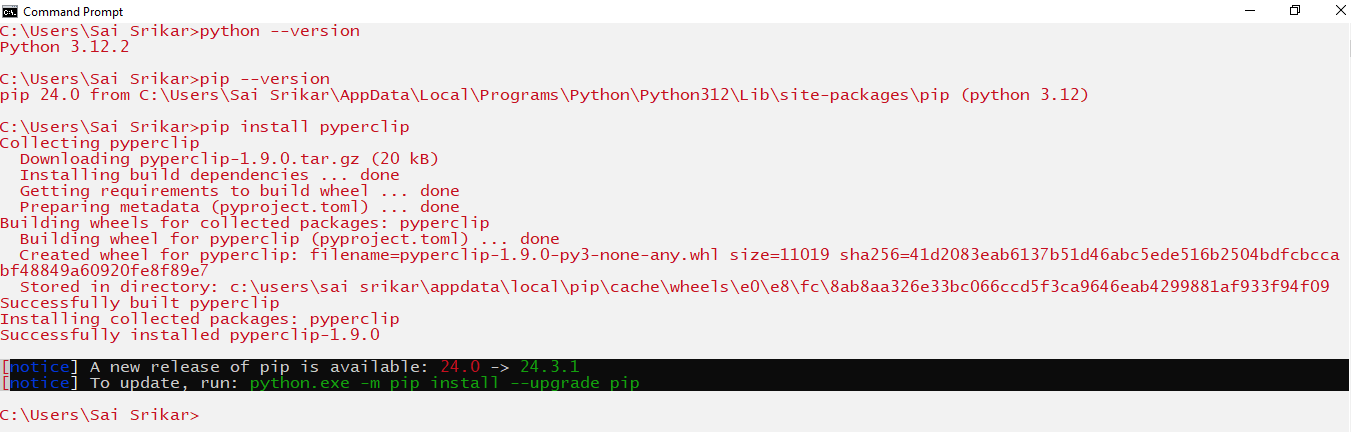
for copy and paste clipboard functions.

==> we can use this in both python2 and Python-3 also.

Install Pyperclip:

==================

$ pip install pyperclip



Note:

====

To update the pip command:

$ python.exe -m pip install --upgrade pip



import pyperclip

text = "Welcome To Srikara IT"

# copying the text

pyperclip.copy(text)

# Paste the text

ctext = pyperclip.paste()

print("Original Text = ",text)

print("Copied Text = ",ctext)

========================================

Random Password Generator

=========================

import random

# import pyperclip

from tkinter import \*

from tkinter.ttk import \*

def low():

entry.delete(0,END)

length = var1.get()

lower = "abcdefghijklmnopqrstuvwxyz"

upper = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"

digits = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789!@#$%^&\*()"

password = ""

if var.get() == 1:

for i in range(0,length):

password = password + random.choice(lower)

return password

elif var.get() == 0:

for i in range(0,length):

password = password + random.choice(upper)

return password

elif var.get() == 3:

for i in range(0,length):

password = password + random.choice(digits)

return password

else:

print("Please select an option")

def generate():

password1 = low()

entry.insert(10,password1)

# def copy1():

# random\_password = entry.get()

# pyperclip.copy(random\_password)

window = Tk()

var = IntVar()

var1 = IntVar()

window.title("Random Password Generator")

Random\_password = Label(window,text = "Password")

Random\_password.grid(row = 0)

entry = Entry(window)

entry.grid(row = 0,column = 1)

c\_label = Label(window, text = "Length")

c\_label.grid(row = 1)

# copy\_button = Button(window,text = "Copy",command = copy1)

# copy\_button.grid(row = 0,column = 2)

generate\_button = Button(window,text = "Generate",command = generate)

generate\_button.grid(row = 0,column = 3)

radio\_low = Radiobutton(window, text = "Low", variable = var,value = 1)

radio\_low.grid(row = 1,column = 2,sticky = 'E')

radio\_middle = Radiobutton(window, text = "Medium",variable = var, value = 0)

radio\_middle.grid(row = 1,column = 3,sticky = 'E')

radio\_strong = Radiobutton(window,text = "Strong",variable = var, value = 3)

radio\_strong.grid(row = 1, column = 4,sticky = 'E')

combo = Combobox(window, textvariable = var1)

combo['values'] = (8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32, "Length")

combo.current(0)

combo.bind("<<ComboboxSelected>>")

combo.grid(row = 1,column = 1)

window.mainloop()

**DAY87:** **TKinter Part\_03**:

Setup:

======

pip install Pyperclip

conda install -c conda-forge Pyperclip

==============================================================

\*\* Random Password Generator \*\*

================================

# Python program to generate random

# password using Tkinter module

import random

import pyperclip

from tkinter import \*

from tkinter.ttk import \*

# Function for calculation of password

def low():

entry.delete(0, END)

# Get the length of password

length = var1.get()

lower = "abcdefghijklmnopqrstuvwxyz"

upper = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"

digits = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789 !@#$%^&\*()"

password = ""

# if strength selected is low

if var.get() == 1:

for i in range(0, length):

password = password + random.choice(lower)

return password

# if strength selected is medium

elif var.get() == 0:

for i in range(0, length):

password = password + random.choice(upper)

return password

# if strength selected is strong

elif var.get() == 3:

for i in range(0, length):

password = password + random.choice(digits)

return password

else:

print("Please choose an option")

# Function for generation of password

def generate():

password1 = low()

entry.insert(10, password1)

# Function for copying password to clipboard

def copy1():

random\_password = entry.get()

pyperclip.copy(random\_password)

# Main Function

# create GUI window

root = Tk()

var = IntVar()

var1 = IntVar()

# Title of your GUI window

root.title("Random Password Generator")

# create label and entry to show

# password generated

Random\_password = Label(root, text="Password")

Random\_password.grid(row=0)

entry = Entry(root)

entry.grid(row=0, column=1)

# create label for length of password

c\_label = Label(root, text="Length")

c\_label.grid(row=1)

# create Buttons Copy which will copy

# password to clipboard and Generate

# which will generate the password

copy\_button = Button(root, text="Copy", command=copy1)

copy\_button.grid(row=0, column=2)

generate\_button = Button(root, text="Generate", command=generate)

generate\_button.grid(row=0, column=3)

# Radio Buttons for deciding the

# strength of password

# Default strength is Medium

radio\_low = Radiobutton(root, text="Low", variable=var, value=1)

radio\_low.grid(row=1, column=2, sticky='E')

radio\_middle = Radiobutton(root, text="Medium", variable=var, value=0)

radio\_middle.grid(row=1, column=3, sticky='E')

radio\_strong = Radiobutton(root, text="Strong", variable=var, value=3)

radio\_strong.grid(row=1, column=4, sticky='E')

combo = Combobox(root, textvariable=var1)

# Combo Box for length of your password

combo['values'] = (8, 9, 10, 11, 12, 13, 14, 15, 16,

17, 18, 19, 20, 21, 22, 23, 24, 25,

26, 27, 28, 29, 30, 31, 32, "Length")

combo.current(0)

combo.bind('<<ComboboxSelected>>')

combo.grid(column=1, row=1)

# start the GUI

root.mainloop()

\*\* Simple Calculator \*\*

========================

# importing Tkinter and math

from tkinter import \*

import math

# calc class

class calc:

def getandreplace(self):

"""replace x with \* and Ã· with /"""

self.expression = self.e.get()

self.newtext=self.expression.replace('/','/')

self.newtext=self.newtext.replace('x','\*')

def equals(self):

"""when the equal button is pressed"""

self.getandreplace()

try:

# evaluate the expression using the eval function

self.value= eval(self.newtext)

except SyntaxError or NameError:

self.e.delete(0,END)

self.e.insert(0,'Invalid Input!')

else:

self.e.delete(0,END)

self.e.insert(0,self.value)

def squareroot(self):

"""squareroot method"""

self.getandreplace()

try:

# evaluate the expression using the eval function

self.value= eval(self.newtext)

except SyntaxError or NameError:

self.e.delete(0,END)

self.e.insert(0,'Invalid Input!')

else:

self.sqrtval=math.sqrt(self.value)

self.e.delete(0,END)

self.e.insert(0,self.sqrtval)

def square(self):

"""square method"""

self.getandreplace()

try:

#evaluate the expression using the eval function

self.value= eval(self.newtext)

except SyntaxError or NameError:

self.e.delete(0,END)

self.e.insert(0,'Invalid Input!')

else:

self.sqval=math.pow(self.value,2)

self.e.delete(0,END)

self.e.insert(0,self.sqval)

def clearall(self):

"""when clear button is pressed,clears the text input area"""

self.e.delete(0,END)

def clear1(self):

self.txt=self.e.get()[:-1]

self.e.delete(0,END)

self.e.insert(0,self.txt)

def action(self,argi):

"""pressed button's value is inserted into the end of the text area"""

self.e.insert(END,argi)

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

"""Constructor method"""

master.title('Calculator')

master.geometry()

self.e = Entry(master)

self.e.grid(row=0,column=0,columnspan=6,pady=3)

self.e.focus\_set() #Sets focus on the input text area

# Generating Buttons

Button(master,text="=",width=11,height=3,fg="blue",

bg="orange",command=lambda:self.equals()).grid(

row=4, column=4,columnspan=2)

Button(master,text='AC',width=5,height=3,

fg="red", bg="light green",

command=lambda:self.clearall()).grid(row=1, column=4)

Button(master,text='C',width=5,height=3,

fg="red",bg="light green",

command=lambda:self.clear1()).grid(row=1, column=5)

Button(master,text="+",width=5,height=3,

fg="blue",bg="orange",

command=lambda:self.action('+')).grid(row=4, column=3)

Button(master,text="x",width=5,height=3,

fg="blue",bg="orange",

command=lambda:self.action('x')).grid(row=2, column=3)

Button(master,text="-",width=5,height=3,

fg="red",bg="light green",

command=lambda:self.action('-')).grid(row=3, column=3)

Button(master,text="Ã·",width=5,height=3,

fg="blue",bg="orange",

command=lambda:self.action('/')).grid(row=1, column=3)

Button(master,text="%",width=5,height=3,

fg="red",bg="light green",

command=lambda:self.action('%')).grid(row=4, column=2)

Button(master,text="7",width=5,height=3,

fg="blue",bg="orange",

command=lambda:self.action('7')).grid(row=1, column=0)

Button(master,text="8",width=5,height=3,

fg="red",bg="light green",

command=lambda:self.action(8)).grid(row=1, column=1)

Button(master,text="9",width=5,height=3,

fg="blue",bg="orange",

command=lambda:self.action(9)).grid(row=1, column=2)

Button(master,text="4",width=5,height=3,

fg="red",bg="light green",

command=lambda:self.action(4)).grid(row=2, column=0)

Button(master,text="5",width=5,height=3,

fg="blue",bg="orange",

command=lambda:self.action(5)).grid(row=2, column=1)

Button(master,text="6",width=5,height=3,

fg="white",bg="blue",

command=lambda:self.action(6)).grid(row=2, column=2)

Button(master,text="1",width=5,height=3,

fg="red",bg="light green",

command=lambda:self.action(1)).grid(row=3, column=0)

Button(master,text="2",width=5,height=3,

fg="blue",bg="orange",

command=lambda:self.action(2)).grid(row=3, column=1)

Button(master,text="3",width=5,height=3,

fg="white",bg="blue",

command=lambda:self.action(3)).grid(row=3, column=2)

Button(master,text="0",width=5,height=3,

fg="white",bg="blue",

command=lambda:self.action(0)).grid(row=4, column=0)

Button(master,text=".",width=5,height=3,

fg="red",bg="light green",

command=lambda:self.action('.')).grid(row=4, column=1)

Button(master,text="(",width=5,height=3,

fg="white",bg="blue",

command=lambda:self.action('(')).grid(row=2, column=4)

Button(master,text=")",width=5,height=3,

fg="blue",bg="orange",

command=lambda:self.action(')')).grid(row=2, column=5)

Button(master,text="?",width=5,height=3,

fg="red",bg="light green",

command=lambda:self.squareroot()).grid(row=3, column=4)

Button(master,text="xÂ²",width=5,height=3,

fg="white",bg="blue",

command=lambda:self.square()).grid(row=3, column=5)

# Driver Code

root = Tk()

obj=calc(root) # object instantiated

root.mainloop()