Harsh Jain

harrrssh@gmail.com

Abstract

!

Python 3 notes

Complete topics with NumPy and Pandas

**Index**

1. History Of Python, Why Python And Real Time App Developing Using Python.
2. Features, Python Execution Environment and Literals/Variable/Identifier.
3. Types of Data Types, Number System conversion and typecasting techniques.

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

History of Python

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

Python was created by Guido Van Rossum and Released in 1991

Python 1.4 released on 25 Oct 1996 to Python 1.6 released on 5 Sep 2000

Python 2.0 released on 16 Oct 2000 to Python 2.7.18 released on 20 Apr 2020

Python 3.0 released on 3 Dec 2008 to till now recently released 3.11.5 on 24Aug23

===========

Why Python

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

Python is powerful... and fast;

plays well with others;

runs everywhere;

is friendly & easy to learn;

is Open Source.

These are some of the reasons people who use Python would rather not use anything else(**According to official documentation of python can be seen on python.org**)

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

Real Time Application developed by Using Python Language

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

With Python Programming language we develop around 22 Real Time applications. They are

1. Web Application Development

* Java PL------Servlets, JSP, JSTL, Spring.
* C#.net   PL-------ASP.NET.
* Python PL----->Django, Flask, Pyramid---Code--Simple

2. Gaming Applications

3. Desktop applications (GUI Applications)

4. Image Processing Applications

5. Business Application Development

6. Audio and Video Based Applications

7. Web Scraping / When Harvesting Applications

8. Data Visualization

9. Data Analysis and Data Analytics

10. Scientific Applications

11. Complex Math calculations.

12. Software Development

13. Operating System Installers

14. CAD and CAM Based Software’s.

15. Embedded System Applications

16. IOT Based Applications

17. Console Based Applications.

18. Language Development

19. Automation of Testing

20. Animation applications

21. Education Programs.

22. Computer Vision.

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

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

Features of Python

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

Features of language are nothing but Services or facilities provided by language developers in the languages and used by programmers for developing real time applications.

Python programming Provides 11 features. They are

1. Simple.

2. Freeware and Open Source.

3. Platform Independent Language.

4. Dynamically Typed Language.

5. Interpreted Programming Language.

6. High Level Programming Language.

7. Both Procedural and Object-Oriented Programming Language.

8. Robust (Strong).

9. Extensible.

10. Embedded.

11. Supports Third Party Modules such as NumPy, Pandas, matplotlib, SciPy and scikit-learn, etc.….

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

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

1. Simple

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

=>Python is one of the simple programming languages because of 3 important Technical Features. They are

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

1. Python Programming Provides "Rich Set of APIs". So that Python programmer can re-use the pre-defined code without writing our code

Definition of API (Application Programing Interface):

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

=>An API is a collection of MODULES.

=>A Module is a collection of Attributes, Functions and Class Names

Examples:- calendar, math, cmath, random, gc, sys .... etc.

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

2. Python Programming Provides In-built facility called "Garbage Collector" and whose role is to collect Un-Used Memory space and Improves the Performance of Python Based Applications.

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

Definition of Garbage Collector:

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

=>A Garbage Collector is one of the In-built programs in python software and which is running in the background of our regular Python Program and whose role is collecting Un-Used Memory space and Improves the Performance of Python Based Applications.

=>Hence Garbage Collector takes care about Automatics Memory Management.

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

3. Python Programming Provides User-Friendly Syntaxes. So that we can develop error-free programs and Limited span of time.

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

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

2. Freeware and Open Source

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

Freeware:

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

=>Any software is said to be Freeware if It can be downloaded freely from official website.

=>Python Software – **python.org and pypi.org for python packages**

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

Open Source:

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

=>The standard Name of Python is called "CPYTHON"

=>Some of company venders came forward and Customized "CPYTHON" and used the customized versions as In-house tools in their companies. Such Type of Customized versions of python are called "Distributions of Python"

=>Some of the Distributions of Python are:-

1) Jpython or Jython(jython.org)---->Used for Running Java Based Applications.

2) Iron Python(ironpython.net)---->Used to Run C#.net applications.

3) Micro Python(micropython.org)--------->Used to develop micro controllers.

4) Ruby Python or rhyton---------->Used to run Ruby Based Applications.

5) Anaconda Python---->Used to Run Hadoop Applications / Big Data Applications.

.....................etc.

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

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

3. Platform Independent

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

=>Platform means type of OS being Used for running or executing the Program.

=>An application or Language is said to be Platform Independent if whose Programs runs on Every Operating System.

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

=>Languages Comparison

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

=>C,CPP...etc.------>Platform Dependent----Data types of C,CPP are size restricted.

=>Java---------------->Platform Independent--objects with limited values

=>Python------------>Platform Independent--objects with unlimited values

(Objects--Data Storage)

=>All values in python are stored in the form of Objects and objects are independent from OS.

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

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

4. Dynamically Typed

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

=>In the context programming languages, we have two types of languages. They are

1. Static Typed Programming Languages.

2. Dynamically Typed Programming Languages.

1. Static Typed Programming Languages.

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

=>In These Programming Languages, Programmer must define a variable with Data Type otherwise we get Compile Time Error.

Examples: int a,b,c;-------Variable Declaration----Mandatory

a=10

b=20

c=a+b

Examples: C,CPP, JAVA , .net.........

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

2. Dynamically Typed Programming Languages.

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

=>In These Programming Languages, Programmer need not to specify the data type of the variables. So that data type of Variable decided by Python Execution Environment based value assigned by Programmer.

Examples:

>>> a=10

>>> b=1.2

>>> c=a+b

>>> print(a, type(a))--------------------- 10 <class 'int'>

>>> print(b, type(b))-------------------- 1.2 <class 'float'>

>>> print(c, type(c))-------------------- 11.2 <class 'float'>

Examples: Python

----------------------------------------------------------------=>Hence All values in Python Programming are stored in the form of Objects.

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

5. Interpreted Programming Language

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

=>When we develop any python program, we must give some file name with an extension .py (FileName.py).

=>When we execute python program, two process taken place internally they are

a) Compilation Process.

b) Execution Process.

=>In COMPILATION PROCESS, the python Source Code submitted to Python Compiler and It reads the source Code, check for errors by verifying syntaxes and if no errors found then Python Compiler Converts this into Intermediate Code called BYTE CODE with an extension .pyc (FileName.pyc). If errors found in source code, then we error displayed on the console.

=>In EXECUTION PROCESS, The PVM reads the Python Intermediate Code(Byte Code) Line by Line and Converted into Machine Understandable Code (Executable or binary Code) and it is read by OS and Processer then finally Gives Result.

=>Hence In Python Program execution, Compilation Process and Execution Process is taking place doing Line by Line conversion and it is one of the Interpretation Based Programming Language.

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

Definition of PVM ( Python Virtual Machine)

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

=>PVM is one program in Python Software and whose role is to read LINE by LINE of Byte Code and Converted into Machine Under stable Code (Executable or binary Code)

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

6. High Level Programming

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

=>In general, we have two types of Programming languages. They are

a) Low Level Programming Languages.

b) High Level Programming Languages.

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

a) Low Level Programming Languages:

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

=>In These Programming Languages, we represent the data in lower-level data like Binary, Octal and Hexadecimal and This type data is not by default untestable by Programmers and end users.

Examples:- a=0b1111110000111101010---binary data

b=0o23-----octal

c=0xface----Hexadecimal

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

b) High Level Programming Languages.

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

In These Programming Languages, even we represent the data in lower-level data like Binary, Octal and Hexadecimal , the High-Level Programming Languages automatically converts into Decimal number System data, which is understandable by Programmers and end-users and python is one High Level Programming Language.

Example : Python

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

7. Both Procedural and Object-Oriented Programming Language

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

Yes, Python support both Object  Oriented and Procedural  Programming language as it is a high-level programming language designed for general purpose programming. Python are multi-paradigm, you can write programs or libraries that are largely procedural, object-oriented, or functional in all these languages. It depends on what you mean by functional. Python does have some features of a functional language.

OOP's concepts like, Class, Encapsulation, Polymorphism, Inheritance etc... in Python makes it as an object-oriented programming language.

In Similar way we can created procedural program through python using loops, for, while etc. ...and control structure.

**Example**

class Rectangle:

def \_\_init\_\_(self, length, breadth, unit\_cost=0):

self.length = length

self.breadth = breadth

self.unit\_cost = unit\_cost

def get\_perimeter(self):

return 2 \* (self.length + self.breadth)

def get\_area(self):

return self.length \* self.breadth

def calculate\_cost(self):

area = self.get\_area()

return area \* self.unit\_cost

# breadth = 120 cm, length = 160 cm, 1 cm^2 = Rs 2000

r = Rectangle(160, 120, 2000)

print("Area of Rectangle: %s cm^2" % (r.get\_area()))

print("Cost of rectangular field: Rs. %s " %(r.calculate\_cost()))

**Output**

Area of Rectangle: 19200 cm^2

Cost of rectangular field: Rs. 38400000

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

8. Robust (Strong )

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

=>Python is one of Robust(Strong) language because of "Exception Handling".

=>Exception:- Runtime Error of the Program are called Exceptions.

=>Exception by default generates Technical Error Messages.

=>Exception Handling:- The process of converting Technical Error Messages into User Friendly Error Messages is called as Exception Handling.

=>If the Python program uses Exception Handling the Python program is Robust.

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

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

9. Extensible

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

=>The Python programming giving its programming facilities to other languages and hence Python is one of the extensible Programming languages.

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

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

10. Embedded

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

=>Python Programming can call other languages coding segments for fastest execution

Example: Python code can call C programming Code.

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

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

11.Extensive Support for Third Party APIs

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

=>As Python Libraries / API can do many tasks and Operations and unable perform complex operations and to solve such complex operations more easily and quickly some of the modules we use as Third-Party APIs such as

1) NumPy----Numerical calculations

2) pandas---Analysis tool

3) matplotlib-----Data Visualization

4) SciPy

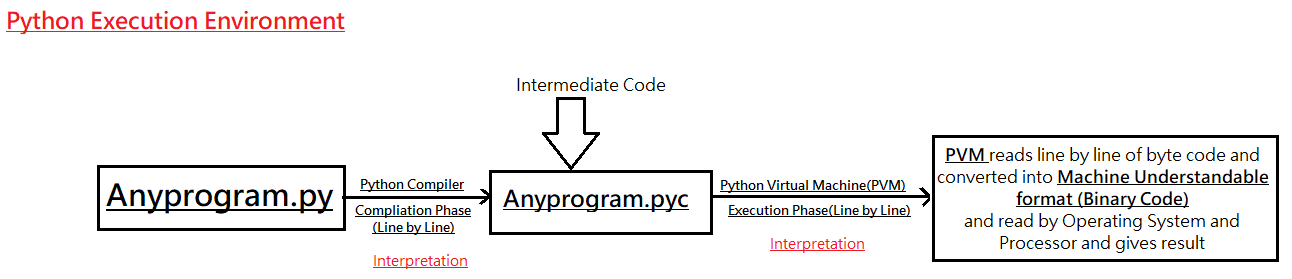
5) scikit

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

Python Execution Environment

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

For Example, we created a program in python language(Anyprogram.py) and when we executed this program during execution python compiler(interpretation) in compiling phase do line by line compilation of program and generates .pyc(anyprogram.pyc) file we often called as intermediate code or byte code now this code is taken by PVM(Python Virtual Machine) and PVM reads line by line of byte code and converted into Machine understandable format code or(Binary Code 0s

and 1s) Now this code can understand by any operating system then gives output. 

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

Literals in Python

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

=>Literals in Python are nothing but Values passing to the Python Program.

=>Programmatically, when we write any python program, we must enter inputs and such inputs are also called Literals or Values.

=>Hence all types of values are called Literals.

=>In Python Programming, we have 5 Types of Literals. They are

1. Integer Literals.

2. Floating Point Literals.

3. Str Literals.

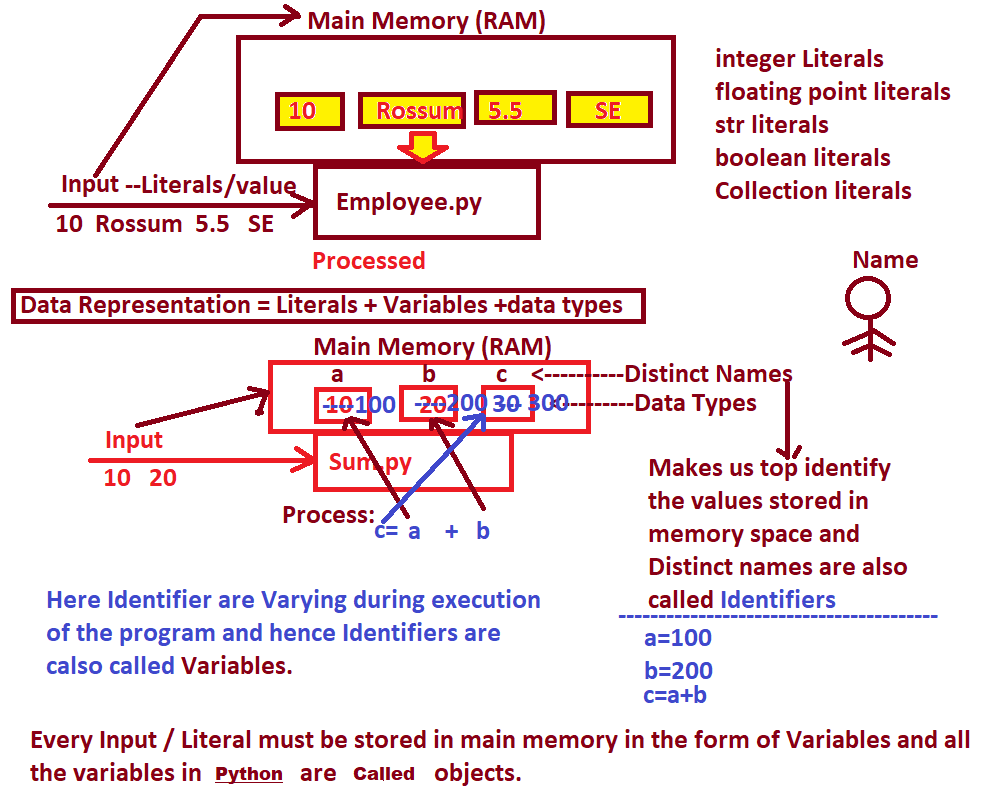
4. Boolean Literals.

5. Collection Literals.

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

Explanation:-

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

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

Identifiers or Variables in Python

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

=>When we enter input or literals, they are stored in main memory by allocating enough memory space with help of data types. To Process the data which was stored in main memory, as a programmer, we must take some distinct names and these names makes us to identify the values stored in memory space and hence names are called "Identifiers". The values of Identifiers are changing or varying during program execution and hence Identifiers are also called "Variables".

=>Hence All types of Input or Literal must be stored in the form of Variables.

=>All Types of Variables are called Object.

=>Definition of Variable:

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

=>A Variable is an Identifier whose value can be changed during execution of the program.

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

=>Rules for Using Identifiers or Variables in Python Program:

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

=>To use Variables in Python Program, we must follow the following rules.

1. The Variable Name is a combination of Alphabets, Digits, and a special Symbol Underscore ( \_ ) only.

2. The First Letter of Variable Name must start either with an Alphabet or Under Score.

Examples:

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

1abc=23-----Invalid

sal123=45---valid

\_sal=4.5---valid

$sal=4.5--invalid

\_123=5.6--valid

\_sal\_=56--valid

3. No Special Symbols are allowed within Variable Name except Under Score

( Note # is used for Commenting in Python )

Examples:

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

sal=45--valid

emp sal=5.6---Invalid

sal#emp=56--valid ( sal is 45 printed)

sal-emp=56--invalid

sal$emp=67--invalid

a#sal=45----Invalid because variable 'a' is not defined earlier(Name Error)

4. No keywords to be used as Variable Names ( Key words are reserved words in programming languages and they give specific meaning to compilers).

Examples:

else=45----invalid because 'else' is key word

else1=56--valid

if=45--invalid because 'if' is key word

\_if=56--valid

int=67-----Valid, because all class names are not keywords

5. All Variable names are Case Sensitive.

Examples:

>>> age=19-----valid

>>> AGE=20---------valid

>>> Age=21---------valid

>>> age=22-----valid

print(age,AGE,Age,aGe)---19 20 21 22

6. It is ways recommended to take user-friendly variable names

Examples:

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

totsalsalaryofanemployee=45-----Valid but not recommended

tot\_sal\_emp=45--valid and recommended.

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

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

Data Types in Python

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

=>The purpose of Data Types in Python is that "To allocate sufficient amount of memory space for storing the Values or Literals in main memory (RAM) of computer".

=>In Python Programming 14 data types(class) and they are classified into 6 categories.

I. Fundamental Category Data Types

* Int
* Float
* Bool
* Complex

II. Sequence Category Data Types

* Str
* Bytes
* Bytearray
* Range

III. List Category Data Types. (Collections Data Types or Data Structures)

* List
* Tuple

Iv. Set Category Data Types (Collections Data Types or Data Structures)

* Set
* Frozenset

V. Dict Category Data Type (Collections Data Types or Data Structures)

* Dict

VI. None Type Category Data Types.

* Nonetype ( Whose Value Is Nothing Called None)

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

I. Fundamental Category data Types

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

=>The purpose of Fundamental Category data Types in Python is that "To Store Single value in single object / variable".

=>In Python Programming, we have 4 data types in Fundamental Category. They are

1. int
2. float
3. bool
4. complex

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

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

1. int

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

=>'int' class is one of the pre-defined class and is considered a fundamental data type.

=>The purpose of 'int' data type is that "To store Integer data or Integral data or Whole Numbers(Numbers without decimal places)."

Examples:

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

Python Instructions Output

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

>>> a=100

>>> b=-345

>>> c=0

>>> print(a,type(a))---------------------------------------100 <class 'int'>

>>> print(b,type(b))---------------------------------------345 <class 'int'>

>>> print(c,type(c))----------------------------------------0 <class 'int'>

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

>>> a=12

>>> b=34

>>> c=a+b

>>> print(a,type(a))---------------------------------------12 <class 'int'>

>>> print(b,type(b))---------------------------------------34 <class 'int'>

>>> print(c,type(c))---------------------------------------46 <class 'int'>

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

>>> a=-12

>>> b=-45

>>> c=a+b

>>> print(a,type(a))---------------------------------------12 <class 'int'>

>>> print(b,type(b))---------------------------------------45 <class 'int'>

>>> print(c,type(c))---------------------------------------57 <class 'int'>

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

=>With 'int' data type, we can also store Different types of Number System Data.

=>In Computer system architecture, we have 4 Types of Number Systems. They are

a) Decimal Number System

b) Binary Number System

c) Octal Number System

d) Hexadecimal Number System.

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

a) Decimal Number System

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

=>It is one of the default number System understandable by people.

=>This Number System contains the following digits.

Digits: 0 1 2 3 4 5 6 7 8 9 ---------------->Total Digits=10

Base : 10

=>All Base 10 Literals are called Decimal Number System Data.

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

b) Binary Number System

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

=>This Number System Data Understandable by OS and Processor during application

Execution.

=>This Number System contains the following digits.

Digits: 0 1 ---------------->Total Digits=2

Base : 2

=>In Python Programming, Binary Data must represent or stored in the variables by preceded with 0b or 0B

=>Syntax:- varname=0b Binary Data

(OR)

varname=0B Binary Data

=>Even we store Binary data, The Python Programming Environment will convert Binary Data into Decimal Value.

Examples: outputs

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

>>> a=0b1110

>>> print(a, type(a))--------------14 <class 'int'>

>>> bin(14)------------------------- '0b1110'

>>> a=0B1111

>>> print(a, type(a))--------------15 <class 'int'>

>>> c=0b1010+0b1111

>>> print(c,type(c))--------------25 <class 'int'>

>>> a=0b101102----------------Syntax Error: invalid digit '2' in binary literal

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

c) Octal Number System

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

=>This Number System Data Used in Micro Processor Programming (8085,8086)

=>This Number System contains the following digits.

Digits: 0 1 2 3 4 5 6 7 ---------------->Total Digits=8

Base : 8

=>In Python Programming, Octal Data must represent or stored in the variables by preceded with 0o or 0O

=>Syntax:- varname=0o Octal Data

(OR)

varname=0O Octal Data

=>Even we store Octal data, The Python Programming Environment will convert Octal Data into Decimal Value.

Examples: Outputs

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

>>> print(a,type(a))-----------23 <class 'int'>

>>> a=0o123

>>> print(a,type(a))----------83 <class 'int'>

>>> oct(83)-------------------'0o123'

>>> oct(23)------------------'0o27'

>>> oct(0b1111)--------------'0o17'

>>> a=0O128-------------------Syntax Error: invalid digit '8' in octal literal

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

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

d) Hexadecimal Number System.

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

=>This Number System Data Used In development of OS

=>This Number System contains the following digits.

Digits: 0 1 2 3 4 5 6 7 8 9

A(10) B(11) C(12) D(13) E(14) F(15)---------------->Total Digits=16

Base : 16

=>In Python Programming, Hexadecimal Data must represent or stored in the variables by preceded with 0x or 0X

=>Syntax:- varname=0x Hexadecimal Data

(OR)

varname=0X Hexadecimal Data

=>Even we store Hexadecimal data, The Python Programming Environment will convert Hexadecimal Data into Decimal Value.

Examples: Outputs

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

>>> a=0xAC

>>> print(a,type(a))--------------172 <class 'int'>

>>> a=0Xac

>>> print(a,type(a))----------------172 <class 'int'>

>>> a=0xBEE

>>> print(a,type(a))----------------3054 <class 'int'>

>>> a=0xBEER-------------Syntax Error: invalid hexadecimal literal

>>> a=0XFACE

>>> print(a,type(a))----------------64206 <class 'int'>

>>> a=0xACC

>>> print(a,type(a))----------------2764 <class 'int'>

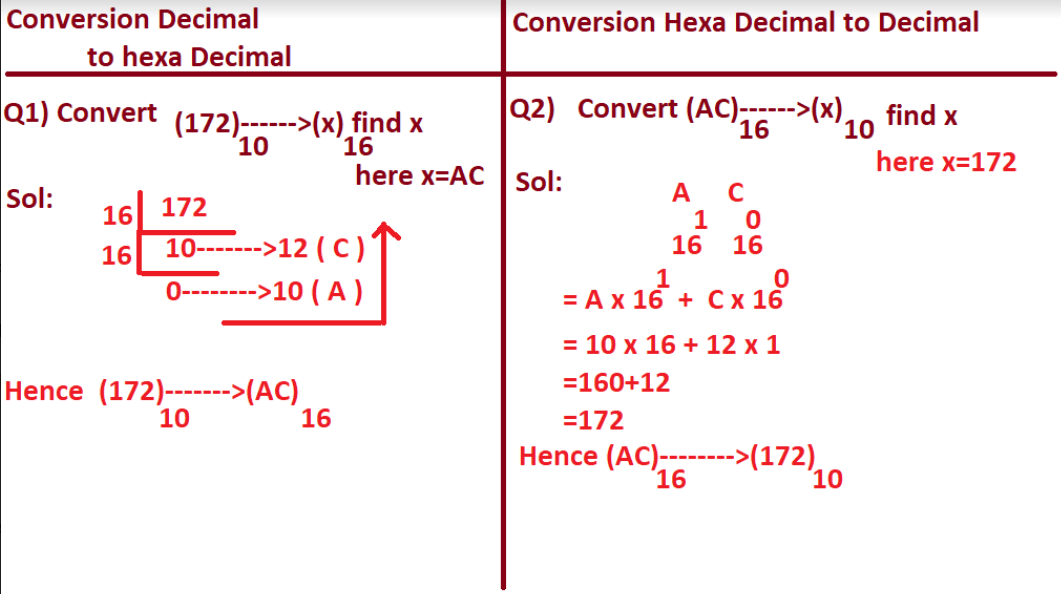
---------------------------------------------------------------------------------

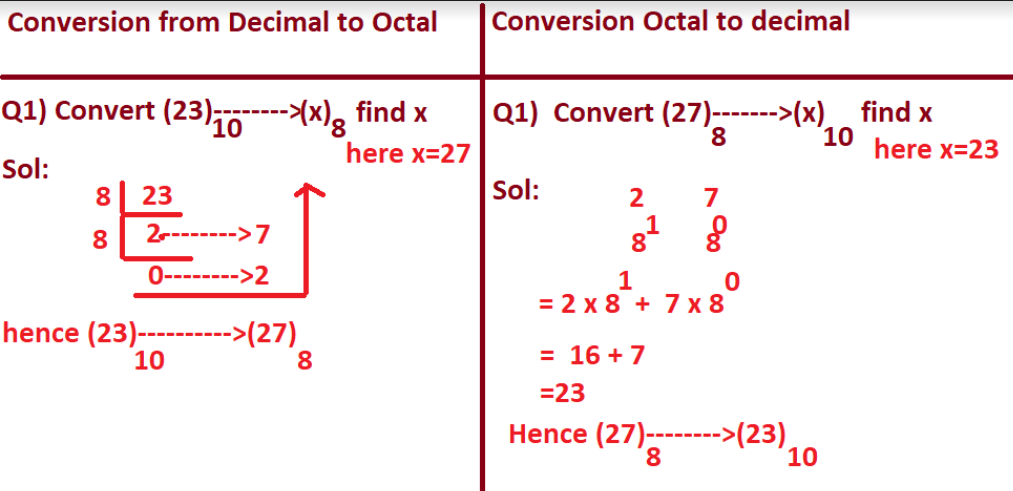
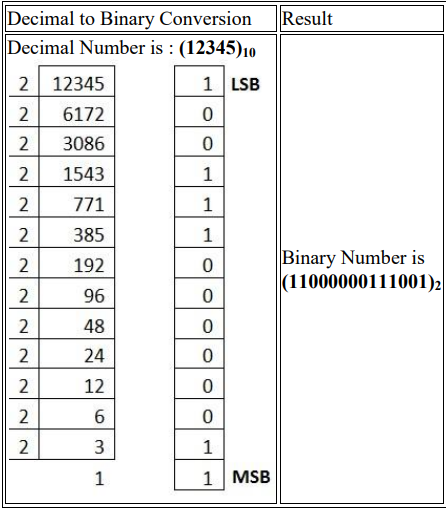
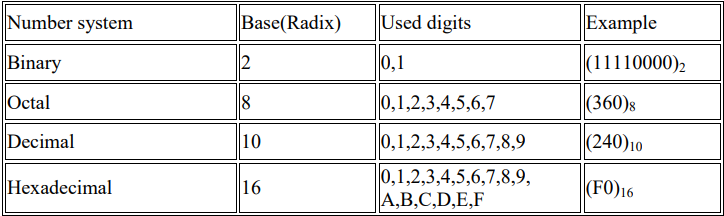
NOTE:

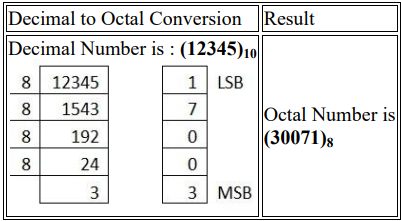
---------------

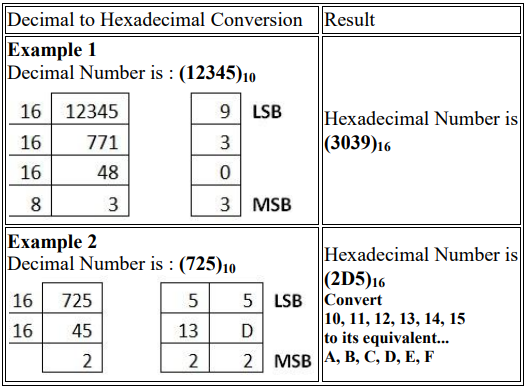
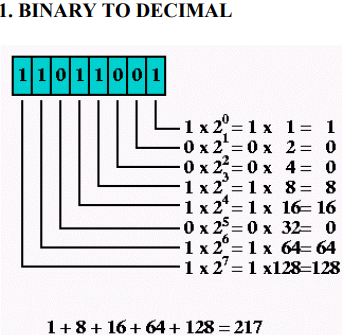
>>> a=0766---------------Syntax Error: leading zeros in decimal integer literals re not permitted; use an 0o(zero and o character) prefix for octal integers

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









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

2. float

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

=>'float' class is one of the pre-defined data class and treated as Fundamental Data Type.

=>The purpose of float data type is that " To Store Real Constant Values or Floating-Point Values(Number with Decimal Places) ".

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

Examples:

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

>>> a=12.34

>>> print(a,type(a))--------------------12.34 <class 'float'>

>>> a=1.2

>>> b=3.4

>>> c=a+b

>>> print(a,type(a))--------------------1.2 <class 'float'>

>>> print(b,type(b))------------------3.4 <class 'float'>

>>> print(c,type(c))-------------------4.6 <class 'float'>

>>> a=-12.34

>>> b=-45.67

>>> c=a+b

>>> print(a,b,c)------------------ -12.34 -45.67 -58.010000000000005

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

=>float data type does not support Binary, Octal and Hexadecimal Number and by default supports only decimal Number only.

Examples:

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

>>> a=0b1010.0b1111---------------Syntax Error: invalid decimal literal

>>> a=0o23.0xACC------------------Syntax Error: invalid decimal literal

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

=>float data type also store scientific notation of floating-point values. The advantage of scientific notation of floating-point value is that "To take less memory space".

Examples:

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

>>> a=3e2

>>> print(a,type(a))-----------------------300.0 <class 'float'>

>>> a=10e-3

>>> print(a,type(a))--------------------- 0.01 <class 'float'>

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

>>> a=0.000000000000000000000000000000000000004

>>> print(a,type(a))------------- 4e-39 <class 'float'>

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

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

3. bool

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

=>"bool" class is one of the pre-defined class and treated as fundamental data type.

=>The purpose of bool data type is that " To Store True/False Values"

=>True and False are called Logical Values.

=>True and False are the keywords and they are treated as values for bool data type.

=>Internally, The Value of True is Considered as 1 and Value of False is Considered as 0.

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

Examples:

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

>>> a=True

>>> print(a,type(a))------------------True <class 'bool'>

>>> a=False

>>> print(a,type(a))-----------------False <class 'bool'>

>>> a=true------------Name Error: name 'true' is not defined. Did you mean: 'True'?

>>> a=false-------Name Error: name 'false' is not defined. Did you mean: 'False'?

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

>>> a=True

>>> b=False

>>> print(a+b)---------------1

>>> print(a+b-a)-------------0

>>> print(a\*2-3\*False)-----------2

>>> a=True

>>> v=False

>>> print(a\*0b1010-v+a)------------11

>>> print(True+True+True\*False)---------2

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

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

4. complex

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

=>'complex' class is one of the pre-defined class and treated as Fundamental data type.

=>The purpose of complex data type is that "To Store Complex Values."

=>The General format of complex Number is shown below.

a+bj or a-bj

=>Here 'a' is called REAL Part

=>Here 'b' is called IMAGINARY Part

=>Here 'j' represents sqrt(-1) or sqr(j)=-1

=>Internally Real and Imaginary parts are treated as float values.

=>Programmatically, to extract the data from complex object, we have two pre-defined attributes. They are

1) real

2) imag

Syntax: complexobj.real---->Gives Real Part of complex object.

complexobj.imag--->Gives Imaginary part of Complex Object.

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

Examples:

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

>>> a=2+3j

>>> print(a, type(a))-------------------------(2+3j) <class 'complex'>

>>> a=2-3j

>>> print(a, type(a))-------------------------(2-3j) <class 'complex'>

>>> a=2.3+4.5j

>>> print(a, type(a))-------------------------(2.3+4.5j) <class 'complex'>

>>> a=23+4.5j

>>> print(a, type(a))----------------------(23+4.5j) <class 'complex'>

>>> a=0+4j

>>> print(a, type(a))----------------------4j <class 'complex'>

>>> a=0+0j

>>> print(a, type(a))----------------------0j <class 'complex'>

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

>>> a=10+4j

>>> print(a, type(a))----------------------------------(10+4j) <class 'complex'>

>>> print(a.real)-------------------------------------10.0

>>> print(a.imag)-----------------------------------4.0

>>> a=-4.5-6.7j

>>> print(a, type(a))--------------------------(-4.5-6.7j) <class 'complex'>

>>> print(a.real)---------------------------- -4.5

>>> print(a.imag)---------------------------- -6.7

>>> a=-6j

>>> print(a, type(a))----------------------- (-0-6j) <class 'complex'>

>>> print(a.real)------------------------------ -0.0

>>> print(a.imag)----------------------------- -6.0

>>> print(a.imaginary)-----------------Attribute Error: 'complex' object has no attribute 'imaginary'

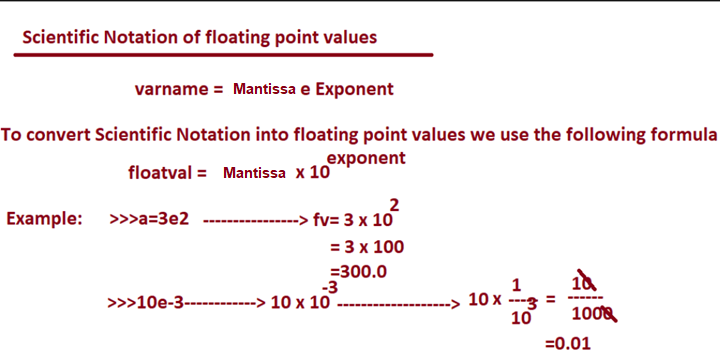
-----------------------------------------------------------------------------------------------------------

>>> (2.3+4j).real-------------------2.3

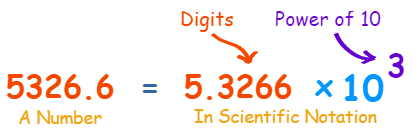
>>> (2.3+4j).imag------------------4.0

>>> (2.3+4j).imag+True---------------5.0

>>> (2.3+4j).imag+True+0b1111-------------20.0

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

Mantissa Having Two Definitions:  
1: The part of a number after the "."  
Example: in 2.71828 the mantissa is 0.71828  
  
2: In scientific notation the mantissa is the digits without the ×10n part.  
Example: in 5.3266 × 103 the mantissa is 5.3266



Where a number is written in two parts:  
• First: the digits, with the decimal point placed after the first digit,  
• Followed by: ×10 to a power that will put the decimal point back where it should be.

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

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

II. Sequence Category data Types

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

=>The purpose of Sequence Category data Types in Python is that to store "Sequence of Values." In memory space.

=>In Python programming, we have 4 Data Types in Sequence Category. They are

1. str

2. bytes

3. bytearray

4. range

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

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

1. str

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

Index

----------

=>Purpose of str

=>Types of strs

=>Notations for storing str data

=>Str Memory Management

=>Operations on str data

a) Indexing

b) Slicing

=>Programming Examples

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

=>'str' is class one of the pre-defined class and treated as Sequence Data Type.

=>The purpose of str data type is that "To store String data or text data or Alphanumeric data or numeric data or any type data within double Quotes or single quotes or triple double quotes and triple single quotes. "

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

=>Definition of str:

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

=>str is a collection of Characters or Alphanumeric data or numeric data or any type data with in double Quotes or single quotes or triple double quotes and triple single quotes. "

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

Types of Str data

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

=>In Python Programming, we have two types of Str Data. They are

1. Single Line String Data

2. Multi Line String Data

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

1. Single Line String Data:

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

=>Syntax1:- varname=" Single Line String Data "

(OR)

=>Syntax2:- varname=' Single Line String Data '

=>With the help of double Quotes (" ") and single Quotes (' ') we can store single line str data only but not possible to multi line string data.

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

2 Multi Line String Data:

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

=>Syntax1:- varname=" " " String Data1

String Data2

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

String data-n " " "

(OR)

=>Syntax2:- varname=' ' ' String Data1

String Data2

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

String data-n ' ' '

=>With the help triple double Quotes (" " " " " ") and Tripple single Quotes (' ' ' ' ' ') we can store single line str data multi line string data.

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

>>> s1="Python Programming"

>>> print(s1,type(s1))-----------------Python Programming <class 'str'>

>>> s2='Java Programming'

>>> print(s2,type(s2))--------------------Java Programming <class 'str'>

>>> addr1="Guido Van Rossum-----------Syntax Error: unterminated string literal (detected at line 1)

>>> addr1='Guido Van Rossum------------Syntax Error: unterminated string literal (detected at line 1)

>>> addr1="""Guido Van Rossum

... FNO:3-4, Red Sea Side

... Python Software Foundation

... Nether Lands-56"""

>>> print(addr1,type(addr1))

Guido Van Rossum

FNO:3-4, Red Sea Side

Python Software Foundation

Nether Lands-56 <class 'str'>

>>> addr2='''James Gosling

... HNO:13-45, Hill Side

... Sun Micro System INC

... USA-567'''

>>> print(addr2,type(addr2))

James Gosling

HNO:13-45, Hill Side

Sun Micro System INC

USA-567 <class 'str'>

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

>>> s3="""Python Programming"""

>>> print(s3,type(s3))--------Python Programming <class 'str'>

>>> s4='''Data Science '''

>>> print(s4,type(s4))-----------Data Science <class 'str'>

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

>>> c1="A"

>>> print(c1,type(c1))--------------A <class 'str'>

>>> c2='A'

>>> print(c2,type(c2))----------A <class 'str'>

>>> c3="""A"""

>>> print(c3,type(c3))--------------A <class 'str'>

>>> c4='''A'''

>>> print(c4,type(c4))----------------A <class 'str'>

>>> s5="Python3.10.5"

>>> print(s5,type(s5))--------------Python3.10.5 <class 'str'>

>>> s6="ABCDabcbd45678#$%^&\*\_kvr"

>>> print(s6,type(s6))-----------ABCDabcbd45678#$%^&\*\_kvr <class 'str'>

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

Operations on str data

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

=>On str Data , we can perform two types of Operations. They are

1. Indexing

2. Slicing

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

1. Indexing

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

=>The process of obtaining single character from given str object is called Indexing.

=>Syntax:- strobj[Index]

=>Here Index can be either +ve or -ve

=>If we enter valid Index then we get a Character from str obj.

=>If we enter invalid Index then we get Index Error.

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

Examples:

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

>>> s="PYTHON"

>>> print(s,type(s))--------------PYTHON <class 'str'>

>>> print(s[0])------------P

>>> print(s[2])------------T

>>> print(s[4])------------O

>>> print(s[5])------------N

>>> print(s[-1])-------------N

>>> print(s[-6])------------P

>>> print(s[-2])----------O

>>> print(s[-3])----------H

>>> print(s[-30])------------IndexError: string index out of range

>>> print(s[2])------------T

>>> print(s[12])------------IndexError: string index out of range

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

>>> "Java"[2]---------------'v'

>>> "Java"[-1]---------------'a'

>>> "Java"[-3]--------------'a'

>>> "Java"[-4]-------------'J'

>>> "Java"[-4+2]-----------'v'

>>> "Java"[True]------------'a'

>>> "Java"[False]----------'J'

>>> "1234"[10-8]-------------'3'

>>> ""[3]-------------IndexError: string index out of range

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

2. Slicing Operations

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

=>The Process of obtaining range of characters or sub-string from given string is called Slicing.

=>We can perform slicing Operations with 5 Syntaxes or 5 types. They are

* strobj[Begin:End]
* strobj[Begin: ]
* strobj[ : End]
* strobj[ : ]
* strobj[Begin:End:Step]

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

Syntax1: strobj[BEGIN INDEX:END INDEX] Hint:-strobj= Str object

=>This syntax obtains range of Characters from strobj from BEGIN INDEX to END INDEX-1 provided BEGIN INDEX<END INDEX otherwise we never get any output.

(space or ' ' is a result)

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

Examples:

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

>>> s="PYTHON"

>>> print(s,type(s))---------------PYTHON <class 'str'>

>>> s[0:4]--------------------'PYTH'

>>> s[3:6]-----------------'HON'

>>> s[2:6]-----------------'THON'

>>> s[4:6]--------------------'ON'

>>> s[3:2]-------------------- ''

>>> s[-6:-3]-----------------'PYT'

>>> s[-4:-1]------------------'THO'

>>> s[-3:-1]------------------'HO'

>>> s[-5:-1]------------------'YTHO'

>>> s[2:-1]---------------------'THO'

>>> s[0:-2]---------------------'PYTH'

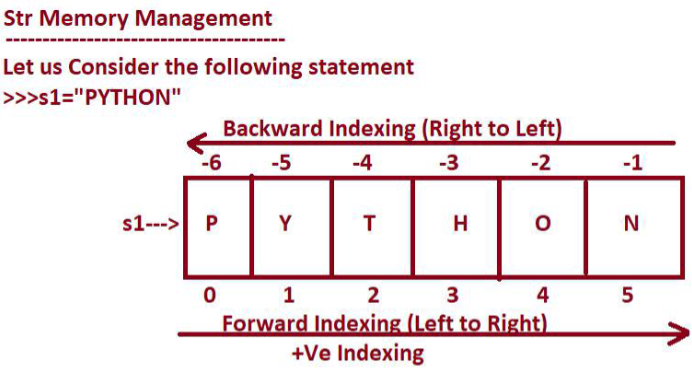
>>> s[2:-2]---------------------'TH'

>>> s[20:-20]----------------''

>>> s[0:-1]--------------------'PYTHO'

>>> s[0:25]-------------------'PYTHON'

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



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

Type Casting Techniques in Python

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

=>The Process of Converting One Possible Type of Value into Another Type of Possible Value is called Type Casting.

=>In Python Programming, We have 5 types of Fundamental Type Casting Techniques. They are.

1. int()

2. float()

3. bool()

4. complex()

5. str()

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

1. int()

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

=>This function is used for converting Possible Type of value into int type value.

=>Syntax: varname=int(float / bool / complex / str )

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

Example: float------>int----->POSSIBLE

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

>>> a=12.34

>>> print(a,type(a))----------------12.34 <class 'float'>

>>> b=int(a)

>>> print(b, type(b))--------------12 <class 'int'>

>>> a=0.9999

>>> print(a,type(a))-------------0.9999 <class 'float'>

>>> b=int(a)

>>> print(b, type(b))------------0 <class 'int'>

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

Example: bool------>int----->POSSIBLE

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

>>> a=True

>>> print(a,type(a))---------------True <class 'bool'>

>>> b=int(a)

>>> print(b, type(b))-------------1 <class 'int'>

>>> a=False

>>> print(a,type(a))--------------False <class 'bool'>

>>> b=int(a)

>>> print(b, type(b))-------------0 <class 'int'>

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

Example: complex------>int----->Not Possible

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

>>> a=10+20j

>>> print(a,type(a))---------------(10+20j) <class 'complex'>

>>> b=int(a)------------TypeError: int() argument must be a string, a bytes-like object, or a real number, not 'complex'

>>> b=int(a.real)

>>> print(b, type(b))-------10 <class 'int'>

>>> b=int(a.imag)

>>> print(b, type(b))----------20 <class 'int'>

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

Example:

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

Case-1:- str int -------------> int--Possible

>>> a="10"

>>> print(a,type(a))------- 10 <class 'str'>

>>> b=int(a)

>>> print(b,type(b))-----------10 <class 'int'>

Case-2 Str float-------> int--- Not Possible

>>> a="12.34"

>>> print(a,type(a))------------->12.34 <class 'str'>

>>> b=int(a)--------ValueError: invalid literal for int() with base 10: '12.34'

Case-3 str bool------------->int---Not Possible

>>> a="True"

>>> print(a,type(a))-------------True <class 'str'>

>>> b=int(a)---------------ValueError: invalid literal for int() with base 10: 'True'

Case-4 str complex--------->int-----Not Possible

>>> a="2+3j"

>>> print(a,type(a))-------2+3j <class 'str'>

>>> b=int(a)------------ValueError: invalid literal for int() with base 10: '2+3j'

Case-5 pure str------------>int-----Not Possible

>>> a="Python"

>>> print(a,type(a))-----------Python <class 'str'>

>>> b=int(a)----------------ValueError: invalid literal for int() with base 10: 'Python'

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

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

2. float()

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

=>This function is used for converting Possible Type of value into float type value.

=>Syntax: varname=float(int / bool / complex / str )

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

Example: int------>float----->Possible

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

>>> a=12

>>> print(a,type(a))-----------------12 <class 'int'>

>>> b=float(a)

>>> print(b,type(b))--------------12.0 <class 'float'>

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

Example: bool------>float----->Possible

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

>>> a=True

>>> print(a,type(a))-----------True <class 'bool'>

>>> b=float(a)

>>> print(b,type(b))------------1.0 <class 'float'>

>>> a=False

>>> print(a,type(a))---------------False <class 'bool'>

>>> b=float(a)

>>> print(b,type(b))---------------0.0 <class 'float'>

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

Example: complex------>float----->Not Possible

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

>>> a=2+3j

>>> print(a,type(a))----------(2+3j) <class 'complex'>

>>> b=float(a)-----------TypeError: float() argument must be a string or a real number, not 'complex'

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

Example:

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

Case-1: str int -------------> float--Possible

>>> a="10"

>>> print(a,type(a))------------- 10 <class 'str'>

>>> b=float(a)

>>> print(b,type(b))------------ 10.0 <class 'float'>

Case-2: Str float-------------> float--->Possible

>>> a="12.34"

>>> print(a,type(a))---------- 12.34 <class 'str'>

>>> b=float(a)

>>> print(b,type(b))---------- 12.34 <class 'float'>

Case-3: str bool------------->float---Not Possible

>>> a="True"

>>> print(a,type(a))----------- True <class 'str'>

>>> b=float(a)----------------ValueError: could not convert string to float: 'True'

Case-4: str complex--------->float--Not Possible

>>> a="2+3j"

>>> print(a,type(a))------------- 2+3j <class 'str'>

>>> b=float(a)-------------- ValueError: could not convert string to float: '2+3j'

Case-5: pure str------------>float---Not Possible

>>> a="Python"

>>> print(a,type(a))--------- Python <class 'str'>

>>> b=float(a)------------- ValueError: could not convert string to float: 'Python'

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

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

3. bool()

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

=>This function is used for converting Possible Type of value into bool type value.

=>Syntax: varname=bool(int / float / complex / str )

=>ALL NON-ZERO VALUES ARE TAKEN AS TRUE

=>ALL ZERO VALUES ARE TAKEN AS FALSE

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

Example1: int------------>bool------>Possible

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

>>> a=123

>>> print(a,type(a))--------------------123 <class 'int'>

>>> b=bool(a)

>>> print(b,type(b))------------------ >True <class 'bool'>

>>> a=-12

>>> print(a,type(a))------------------12 <class 'int'>

>>> b=bool(a)

>>> print(b,type(b))-----------------True <class 'bool'>

>>> a=0

>>> print(a,type(a))----------------0 <class 'int'>

>>> b=bool(a)

>>> print(b,type(b))------------------False <class 'bool'>

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

Example2: float------------>bool------>Possible

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

>>> a=1.2

>>> print(a,type(a))-----------------1.2 <class 'float'>

>>> b=bool(a)

>>> print(b,type(b))------------------True <class 'bool'>

>>> a=0.0000000000000000000000000000000000000000000001

>>> print(a,type(a))--------------1e-46 <class 'float'>

>>> b=bool(a)

>>> print(b,type(b))---------------True <class 'bool'>

>>> a=0.0

>>> print(a,type(a))-----------------0.0 <class 'float'>

>>> b=bool(a)

>>> print(b,type(b))----------------------False <class 'bool'>

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

Example3: complex------------>bool------>Possible

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

>>> a=2+3j

>>> print(a,type(a))----------------------(2+3j) <class 'complex'>

>>> b=bool(a)

>>> print(b,type(b))----------------------True <class 'bool'>

>>> a=0+0j

>>> print(a,type(a))---------------------0j <class 'complex'>

>>> b=bool(a)

>>> print(b,type(b))---------------------False <class 'bool'>

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

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

Example4: str------------------>bool------>Possible

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

>>> a="123"

>>> print(a,type(a))

123 <class 'str'>

>>> b=bool(a)

>>> print(b,type(b))

True <class 'bool'>

>>> a="0"

>>> print(a,type(a))

0 <class 'str'>

>>> b=bool(a)

>>> print(b,type(b))

True <class 'bool'>

>>> a="1.2"

>>> print(a,type(a))

1.2 <class 'str'>

>>> b=bool(a)

>>> print(b,type(b))

True <class 'bool'>

>>> a="0.0"

>>> print(a,type(a))

0.0 <class 'str'>

>>> b=bool(a)

>>> print(b,type(b))

True <class 'bool'>

>>> a="2+3j"

>>> print(a,type(a))

2+3j <class 'str'>

>>> b=bool(a)

>>> print(b,type(b))

True <class 'bool'>

>>> a="Python"

>>> print(a,type(a))

Python <class 'str'>

>>> b=bool(a)

>>> print(b,type(b))

True <class 'bool'>

>>> a=" "

>>> print(a,type(a))

<class 'str'>

>>> b=bool(a)

>>> print(b,type(b))

True <class 'bool'>

>>> a=""

>>> print(a,type(a))

<class 'str'>

>>> b=bool(a)

>>> print(b,type(b))

False <class 'bool'>

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

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

4. complex()

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

=>This function is used for converting Possible Type of value into complex type value.

=>Syntax: varname=complex(/ int / float / bool / str )

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

Example: int------>complex----->Possible

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

>>> a=10

>>> print(a,type(a))---------------10 <class 'int'>

>>> b=complex(a)

>>> print(b,type(b))------------(10+0j) <class 'complex'>

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

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

Example: float------>complex----->Possible

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

>>> a=2.3

>>> print(a,type(a))--------------2.3 <class 'float'>

>>> b=complex(a)

>>> print(b,type(b))--------------(2.3+0j) <class 'complex'>

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

Example: bool------>complex----->Possible

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

>>> a=True

>>> print(a,type(a))---------------True <class 'bool'>

>>> b=complex(a)

>>> print(b,type(b))------------(1+0j) <class 'complex'>

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

Example:

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

Case-1:

>>> a="10" # Str Int-----complex--Possible

>>> print(a,type(a))--------10 <class 'str'>

>>> b=complex(a)

>>> print(b,type(b))---------(10+0j) <class 'complex'>

Case-2:

>>> a="1.2" # Str float-----complex--Possible

>>> print(a,type(a))-----------1.2 <class 'float'>

>>> b=complex(a)

>>> print(b,type(b))----------(1.2+0j) <class 'complex'>

Case-3:

>>> a="True" # str bool-----complex--Not Possible

>>> print(a,type(a))-----------------True <class 'str'>

>>> b=complex(a)-------------------ValueError: complex() arg is a malformed string

Case-4:

>>> a="Python" # Pure Str -----complex--Not Possible

>>> print(a,type(a))--------Python <class 'str'>

>>> b=complex(a)--------ValueError: complex() arg is a malformed string

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

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

5. str()

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

=>This function is used for converting all Types of values into str type value.

=>Syntax: varname=str(int / float / bool / complex )

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

Examples:

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

>>> a=100

>>> print(a,type(a))-----------100 <class 'int'>

>>> b=str(a)

>>> print(b,type(b))------------100 <class 'str'>

>>> b------------------------'100'

>>> a=12.34

>>> print(a,type(a))----------------12.34 <class 'float'>

>>> b=str(a)

>>> print(b,type(b))----------------12.34 <class 'str'>

>>> b-----------------------------'12.34'

>>> a=True

>>> print(a,type(a))-------------True <class 'bool'>

>>> b=str(a)

>>> print(b,type(b))---------------True <class 'str'>

>>> b-------------'True'

>>> a=2-3.5j

>>> print(a,type(a))-----------------(2-3.5j) <class 'complex'>

>>> b=str(a)

>>> print(b,type(b))---------------(2-3.5j) <class 'str'>

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

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

Mutable and Immutable Objects

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

=>Mutable Object:

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

=>A Mutable object is one , whose content can be changed at same address.

Examples: list, set, dict, bytearray.

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

=>Immutable Object:

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

=>An Immutable object is one , which will satisfy the following properties.

a) Values can be modified and modified values placed different Address.

b) Does not support item assignment.

Examples: int, float , bool, complex, str ,bytes, set, tuple.

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

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

2. bytes

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

=>"bytes" class is one of the pre-defined class and treated as Sequence Data Type.

=>The purpose of bytes data type is "To Organize the data the in form of Positive Numerical Integer values in the range of 0 to 256 but can use only 0 to 255".

=>This Data Type is useful in sending the data in the form Cipher Text (Encrypted Format) between multiple Remote machines.

=>An object of bytes is one of the immutable objects.

=>On the object bytes, we can perform Both Indexing and Slicing Operations.

=>Programmatically, we don't have any symbolic Notation for organizing the data for bytes data type. But we can convert other Possible data type value into bytes data type value by using bytes().

>>>varname=bytes(object)

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

Examples:

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

>>> tp=(10,20,30,40,256,23)

>>> print(tp, type(tp))-----------------(10, 20, 30, 40, 256, 23) <class 'tuple'>

>>> b=bytes(tp)----------ValueError: bytes must be in range(0, 256)

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

>>> tp=(10,-20,30,40,255,23)

>>> print(tp, type(tp))-------------(10, -20, 30, 40, 255, 23) <class 'tuple'>

>>> b=bytes(tp)-------------------ValueError: bytes must be in range(0, 256)

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

>>> tp=(10,20,30,40,255,23)

>>> print(tp, type(tp))----------------(10, 20, 30, 40, 255, 23) <class 'tuple'>

>>> b=bytes(tp)

>>> print(b,type(b),id(b))-----------------b'\n\x14\x1e(\off\x17' <class 'bytes'> 2073095630944

>>> for v in b:

... print(v)

...

10

20

30

40

255

23

>>> b[0]----------------------------10

>>> b[-1]---------------------------23

>>> b[0]=34-----------------------TypeError: 'bytes' object does not support item assignment

>>> b[0:4]---------------------b'\n\x14\x1e('

>>> for v in b[0:4]:

... print(v)

...

10

20

30

40

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

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

3.bytearray

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

=>"bytearray" class is one of the pre-defined classes and treated as Sequence Data Type.

=>The purpose of bytearray data type is "To Organize the data the in form of Positive Numerical Integer values in the range of 0 to 256 can use 0 to 255".

=>This Data Type is useful in sending the data in the form Cipher Text (Encrypted Format) between multiple Remote machines.

=>An object of bytearray is one of the mutable objects.

=>On the object bytearray, we can perform Both Indexing and Slicing Operations.

=>Programmatically, we don't have any symbolic Notation for organizing the data for bytearray data type. But we can convert other Possible data type value into bytearray data type value by using bytearray().

varname=bytearray(object)

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

Note:- The Functionality of bytearray is exactly like bytes but an object byte belongs to immutable where as an object bytearray belongs to mutable.

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

Examples:

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

>>> l1=[10,20,30,40,50,256]

>>> print(l1)---------------------------[10, 20, 30, 40, 50, 256]

>>> b1=bytearray(l1)---------------ValueError: byte must be in range(0, 256)

>>> l1=[-10,20,30,40,50,255]

>>> print(l1)---------------------------[-10, 20, 30, 40, 50, 255]

>>> b1=bytearray(l1)----------------ValueError: byte must be in range(0, 256)

>>> l1=[10,20,0,30,40,50,255]

>>> print(l1)-------------------------[10, 20, 0, 30, 40, 50, 255]

>>> b1=bytearray(l1)

>>> print(b1,type(b1))-------------bytearray(b'\n\x14\x00\x1e(2\xff') <class 'bytearray'>

>>> for v in b1:

... print(v)

...

10

20

0

30

40

50

255

>>> for v in b1:

... print(v,end=" ")------------10 20 0 30 40 50 255

>>> b1[0]----------------------------10

>>> b1[-1]---------------------------255

>>> for v in b1[2:5]:

... print(v)

...

0

30

40

>>> print(b1,type(b1),id(b1))------bytearray(b'\n\x14\x00\x1e(2\xff') <class 'bytearray'> 2552260633904

>>> b1[-1]=256-----------------ValueError: byte must be in range(0, 256)

>>> b1[-1]=253

>>> print(b1,type(b1),id(b1))----bytearray(b'\n\x14\x00\x1e(2\xfd') <class 'bytearray'> 2552260633904

>>> for val in b1:

... print(val)

...

10

20

0

30

40

50

253

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

>>> b2=bytes(b1)

>>> print(b2,type(b2),id(b2))------b'\n\x14\x00\x1e(2\xfd' <class 'bytes'> 2552257048624

>>> for val in b2:

... print(val)

>>> ...

10

20

0

30

40

50

253

>>> b2[-1]---------------------------253

>>> b2[-1]=234----------------TypeError: 'bytes' object does not support item assignment

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

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

range

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

=>'range' class is one of the pre-defined classes and treated as Sequence Data Type.

=>The purpose of range data type is that " To store Sequence of Numerical Integer Values by “maintaining Equal Interval of Value."

=>An object of range data type is one of the immutable.

=>On the object of range, we can perform Indexing and Slicing Operations.

=>range data type does not contain any symbolic Notation for storing range of values.

=>range data type contains 3 syntaxes. They are

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

=>Syntax1: varname=range(Value)

=>This Syntax Generates range of values from 0 to Value-1

Examples:

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

>>> r=range(10)

>>> print(r, type(r))

range(0, 10) <class 'range'>

>>> for v in r:

... print(v)

...

0

1

2

3

4

5

6

7

8

9

>>> for val in range(6):

... print(val)

...

0

1

2

3

4

5

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

Syntax2:- varname=range(Begin, End)

=>This syntax generates range of values from Begin to End-1 values

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

Examples:

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

>>> r=range(50,56)

>>> print(r,type(r))---------------range(50, 56) <class 'range'>

>>> for val in r:

... print(val)

...

50

51

52

53

54

55

>>> for val in range(100,104):

... print(val)

...

100

101

102

103

>>> r=range(100,104)

>>> r[0]-------------------100

>>> r[-1]----------------103

>>> for v in r[1:3]:

... print(v)

...

101

102

>>> r[0]=110---------------------TypeError: 'range' object does not support item assignment

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

Syntax3:-varname=range(Begin, End, Step)

=>This syntax generates range of values from Begin to End by maintaining Equal Interval of Value based on value of Step.

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

Examples:

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

>>> r=range(10,21,2)

>>> print(r,type(r))----------------range(10, 21, 2) <class 'range'>

>>> for val in r:

... print(val)

...

10

12

14

16

18

20

>>> for val in range(100,50,-10):

... print(val)

...

100

90

80

70

60

>>> for val in range(100,49,-10):

... print(val)

...

100

90

80

70

60

50

==========

Examples:-

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

Q1) 0 1 2 3 4 5 6 7 8 9----------range(10)

>>> for v in range(10):

... print(v)

...

0

1

2

3

4

5

6

7

8

9

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

Q2) 10 11 12 13 14 15---------------range(10,16)

>>> for s in range(10,16):

... print(s)

...

10

11

12

13

14

15

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

Q3) 1000 1001 1002 1003 1004 1005---range(1000,1006)

>>> for s in range(1000,1006):

... print(s)

...

1000

1001

1002

1003

1004

1005

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

Q4) -1 -2 -3 -4 -5 -6 -7 -8 -9 -10-------range(-1,-11,-1)

>>> for s in range(-1,-11,-1):

... print(s)

...

-1

-2

-3

-4

-5

-6

-7

-8

-9

-10

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

Q5) -1000 -1001 -1002 -1003 -1004 -1005---range(-1000,-1006,-1)

>>> for k in range(-1000,-1006,-1):

... print(k)

...

-1000

-1001

-1002

-1003

-1004

-1005

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

Q6) 10 9 8 7 6 5 4 3 2 1----range(10,0,-1)

>>> for v in range(10,0,-1):

... print(v)

...

10

9

8

7

6

5

4

3

2

1

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

Q7) -100 -99 -98 -97 -96 -95-------range(-100,-94,1) or range(-100,-94)

>>> for k in range(-100,-94):

... print(k)

...

-100

-99

-98

-97

-96

-95

>>> for k in range(-100,-94,1):

... print(k)

...

-100

-99

-98

-97

-96

-95

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

Q8) 10 20 30 40 50 60 70 80 90 100-------range(10,101,10)

>>> for k in range(10,101,10):

... print(k)

...

10

20

30

40

50

60

70

80

90

100

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

Q9) 200 150 100 50 0-------range(200,-1,-50)

>>> for k in range(200,-1,-50):

... print(k)

...

200

150

100

50

0

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

Q10) -5 -4 -3 -2 -1 0 1 2 3 4 5---range(-5,6)

>>> for k in range(-5,6):

... print(k)

...

-5

-4

-3

-2

-1

0

1

2

3

4

5

>>> for k in range(-5,6,1):

... print(k)

...

Output:- -5

-4

-3

-2

-1

0

1

2

3

4

5

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

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

III. List Category data Types. (Collections Data Types or Data Structures )

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

=>The purpose of List Category data Types is that "To store multiple values either of Same Type or Different Type or Both the types in single Object with Unique and Duplicates."

=>List Category contains two data types They are

* list (mutable)
* tuple (immutable)

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

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

1) list

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

Index

---------

=>Properties of List

=>Operations on list

a) Indexing

b) Slicing

=>Pre-defined Functions in list

=>Inner or Nested List

=>Pre-defined Functions in inner list

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

Properties of List:

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

=>'list' class is one of the pre-defined classes and treated as list data type.

=>The purpose of list data type is that "To store multiple values either of Same Type or

Different Type or Both the types in single Object with Unique and Duplicates."

=>The elements of list must be enclosed within Square Brackets [ ] and elements must separate by comma.

=>An object of list maintains Insertion Order. i.e., In whichever Order, we insert the data, in the same order we get the output.

=>On the object list we can perform Both Indexing and Slicing Operations.

=>An object of list belongs to mutable.

=>We can convert any type of value into list type of values by using list()

Syntax: listobj=list(object)

=>We can create two types of list objects. They are

1) empty list

2) non-empty list

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

=>Empty List

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

=>An empty list is one, which does not contain any elements and whose length=0

=>Syntax:- listobj=[ ]

(OR

listobj=list()

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

=>Non-Empty List

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

=>A non-empty list is one, which contains elements and whose length>0

=>Syntax:- listobj=[Val1,Val2.....Val-n ]

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

Examples:

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

>>> l1=[10,20,30,-23,45,0,-45,10,20,456]

>>> print(l1,type(l1))

[10, 20, 30, -23, 45, 0, -45, 10, 20, 456] <class 'list'>

>>> l2=[10,"Rossum",23.45,"Python",True,2+3j]

>>> print(l2,type(l2))

[10, 'Rossum', 23.45, 'Python', True, (2+3j)] <class 'list'>

>>> l2[0]

10

>>> l2[1]

'Rossum'

>>> l2[-1]

(2+3j)

>>> l1[-1]

456

>>> l2=[10,"Rossum",23.45,"Python",True,2+3j]

>>> l2[2:5]

[23.45, 'Python', True]

>>> l2[1:]

['Rossum', 23.45, 'Python', True, (2+3j)]

>>> l2[:3]

[10, 'Rossum', 23.45]

>>> l2[::2]

[10, 23.45, True]

>>> l2[::-1]

[(2+3j), True, 'Python', 23.45, 'Rossum', 10]

>>> l1=[10,20,30,-23,45,0,-45,10,20,456]

>>> print(l1,type(l1),id(l1))

[10, 20, 30, -23, 45, 0, -45, 10, 20, 456] <class 'list'> 2386968938432

>>> l1[1]=200

>>> print(l1,type(l1),id(l1))

[10, 200, 30, -23, 45, 0, -45, 10, 20, 456] <class 'list'> 2386968938432

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

>>> l1=[10,20,30,-23,45,0,-45,10,20,456]

>>> print(l1)---------------------------[10, 20, 30, -23, 45, 0, -45, 10, 20, 456]

>>> len(l1)-----------------10

>>> l2=[ ]

>>> print(l2,type(l2))------------ [] <class 'list'>

>>> len(l2)--------------0

>>> l3=list()

>>> print(l3,type(l3))-----------------[] <class 'list'>

>>> len(l3)-----------------------0

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

>>> l1=[10,20,30,"Ram","Rossum","Rajesh",23.45,-45.67]

>>> print(l1)

[10, 20, 30, 'Ram', 'Rossum', 'Rajesh', 23.45, -45.67]

>>> for x in l1:

... print(x)

...

10

20

30

Ram

Rossum

Rajesh

23.45

-45.67

>>> for x in l1[::2]:

... print(x)

...

10

30

Rossum

23.45

>>> for x in l1[::-2]:

... print(x)

...

-45.67

Rajesh

Ram

20

>>> l1[-1]---------------------45.67

>>> l1[2]----------------------30

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

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

Operations on list

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

Indexing:- we can perform indexing operation on any list whose value is greater than zero Using indexing we can access any element. If we access value greater or equal to the length of list then it shows indexerror.

Example :-

>>>l=['hello', 12, 211.2, 'bn']

>>>l[0]

‘hello’

>>>l[1]

12

>>>l[2]

211.2

>>>l[3]

'bn'

>>>len(l)

4 here length of list is 4

>>>l[5]

Traceback (most recent call last):File "<pyshell#10>", line 1, in <module> l[5]

IndexError: list index out of range

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

Slicing:- On the elements of list we can also perform slicing operation using slicing we can achieve sub-list or small list from main list.

On a list we can perform following type of slicing operations.

* List[start:]
* List[start:stop]
* List[start:stop:step]
* List[:stop]
* List[:stop:step]
* List[::step]

Example:-

>>> l=[1,2.4,"hello",'hi']

>>> l[2:]

['hello', 'hi']

>>> l[1:]

[2.4, 'hello', 'hi']

>>> l[0:]

[1, 2.4, 'hello', 'hi']

>>> l[0:1]

[1]

>>> l[1:2]

[2.4]

>>> l[2:3]

['hello']

>>> l[0:3]

[1, 2.4, 'hello']

>>> l[0:4]

[1, 2.4, 'hello', 'hi']

>>> l[0:3]

[1, 2.4, 'hello']

>>> l[0:1:2]

[1]

>>> l[0:2:3]

[1]

>>> l[1:2:3]

[2.4]

>>> l[2:3:4]

['hello']

>>> l[4:-3:-2]

['hi']

>>> l[::-2]

['hi', 2.4]

>>> l[::-1]

['hi', 'hello', 2.4, 1]

>>> l[::-3]

['hi', 1]

>>> l[1::-3]

[2.4]

>>> l[2::-3]

['hello']

>>> l[4::-3]

['hi', 1]

>>> l[1:2:-3]

[]

>>> l[3:2:-3]

['hi']

>>> l[-3]

2.4

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

Pre-defined Functions in list

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

=>Along with the operations of Indexing and Slicing on list, we can also perform other different Operations by using pre-defined functions present in list object.

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

1. append():

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

=>This Function is used for adding the elements to the list at the end of existing elements of list.

=>Syntax:- listobj.append(Element)

Examples:

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

>>> l1=[10,"Rossum"]

>>> print(l1,id(l1))

[10, 'Rossum'] 2386964774208

>>> l1.append(23.45)

>>> print(l1,id(l1))

[10, 'Rossum', 23.45] 2386964774208

>>> l1.append("PYTHON")

>>> l1.append(True)

>>> print(l1,id(l1))

[10, 'Rossum', 23.45, 'PYTHON', True] 2386964774208

>>> l2=list()

>>> print(l2,id(l2))

[] 2386968945984

>>> l2.append(12.34)

>>> l2.append(23.45)

>>> l2.append(345)

>>> l2.append("Python")

>>> print(l2,id(l2))

[12.34, 23.45, 345, 'Python'] 2386968945984

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

2. insert():

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

=>This Function is used for Inserting An element at Specified Valid Existing Index.

=>Syntax:- listobj.insert(index,Element)

Examples:

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

>>> l1=[10,"Rossum"]

>>> print(l1,id(l1))

[10, 'Rossum'] 2386968946368

>>> l1.append(23.45)

>>> print(l1,id(l1))

[10, 'Rossum', 23.45] 2386968946368

>>> l1.insert(1,"Python")

>>> print(l1,id(l1))

[10, 'Python', 'Rossum', 23.45] 2386968946368

>>> l1.insert(2,234)

>>> print(l1,id(l1))

[10, 'Python', 234, 'Rossum', 23.45] 2386968946368

>>> l1.insert(20,"KVR")

>>> print(l1,id(l1))-----------[10, 'Python', 234, 'Rossum', 23.45, 'KVR'] 2386968946368

>>> l1[20]--------------IndexError: list index out of range

>>> l1.insert(300,"PYTHON")

>>> l1[300]-------------------------IndexError: list index out of range

>>> l1-----------[10, 'Python', 234, 'Rossum', 23.45, 'KVR', 'PYTHON']

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

3. remove(): (Based on Value)

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

=>This function is used for removing the First Occurrence of the specified Value from list object.

=>If the specified value does not exist in list object then we get ValueError.

=>Syntax: listobj.remove(element)

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

Examples:

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

>>> l1=[10,"Rossum","Python",23.45,True]

>>> print(l1,id(l1))

[10, 'Rossum', 'Python', 23.45, True] 2386964774208

>>> l1.remove("Rossum")

>>> print(l1,id(l1))

[10, 'Python', 23.45, True] 2386964774208

>>> l1.remove(True)

>>> print(l1,id(l1))

[10, 'Python', 23.45] 2386964774208

>>> l1.remove(100)----------------------ValueError: list.remove(x): x not in list

>>> l1=[10,20,30,10,20,40,10]

>>> print(l1,id(l1))

[10, 20, 30, 10, 20, 40, 10] 2386968946368

>>> l1.remove(10)

>>> print(l1,id(l1))

[20, 30, 10, 20, 40, 10] 2386968946368

>>> l1.remove(10)

>>> print(l1,id(l1))

[20, 30, 20, 40, 10] 2386968946368

>>> l1.remove(10)

>>> print(l1,id(l1))

[20, 30, 20, 40] 2386968946368

>>> l1.remove(10)---------------ValueError: list.remove(x): x not in list

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

4. pop(index)

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

=>This function is used for removing the elements from list based on Index.

=>Syntax:- listobj.pop(Index)

=>Here Index can be either +ve or -ve.

=>If the index is invalid then we get IndexError.

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

Examples:

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

>>> l1=[10,20,30,40,10,20,10,50]

>>> print(l1)

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

>>> l1.pop(4)

10

>>> print(l1)

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

>>> l1.pop(0)

10

>>> print(l1)

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

>>> l1.pop(4)

10

>>> print(l1)

[20, 30, 40, 20, 50]

>>> l1.pop(10)---------------------IndexError: pop index out of range

Special Examples:

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

>> l1=[]

>>> l1.remove(10)

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

ValueError: list.remove(x): x not in list

>>> l1.pop(2)

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

IndexError: pop from empty list

>>> list().pop(0)

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

IndexError: pop from empty list

>>> list().remove(0)

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

ValueError: list.remove(x): x not in list

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

5. pop():

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

=>This function is used for Removing Last Elements (Last Indexed Element) from list.

=>If we call pop() upon empty list object then we get IndexError.

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

Examples:

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

>>> l1=[10,20,30,40,10,20,10,50]

>>> print(l1)

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

>>> l1.pop()

50

>>> print(l1)

[10, 20, 30, 40, 10, 20, 10]

>>> l1.pop()

10

>>> print(l1)

[10, 20, 30, 40, 10, 20]

>>> l1.pop()

20

>>> print(l1)

[10, 20, 30, 40, 10]

>>> l1.pop()

10

>>> print(l1)

[10, 20, 30, 40]

>>> l1.pop()

40

>>> print(l1)

[10, 20, 30]

>>> l1.pop()

30

>>> print(l1)

[10, 20]

>>> l1.pop()

20

>>> print(l1)

[10]

>>> l1.pop()

10

>>> print(l1)

[]

>>> l1.pop()

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

IndexError: pop from empty list

>>> list().pop()

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

IndexError: pop from empty list

>>> [].pop()

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

IndexError: pop from empty list

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

6.del operator:

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

=>'del' operator is used for removing element(s) either based in Indexing and Slicing.

=>'del' operator can remove even entire object.

=>Syntax1: del listobj[Index]

del listobj[Begin:End:Step]

del listobj

Example:

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

>>> l1=[10,20,30,40,50,10,20]

>>> print(l1,type(l1),id(l1))

[10, 20, 30, 40, 50, 10, 20] <class 'list'> 2382558920000

>>> del l1[0]

>>> print(l1,type(l1),id(l1))

[20, 30, 40, 50, 10, 20] <class 'list'> 2382558920000

>>> del l1[2:5]

>>> print(l1,type(l1),id(l1))

[20, 30, 20] <class 'list'> 2382558920000

>>> del l1

>>> print(l1,type(l1),id(l1))----------NameError: name 'l1' is not defined

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

>>> l1=[10,20,30,40]

>>> l1.remove(10)

>>> l1.remove(20)

>>> l1.remove(30)

>>> l1

[40]

>>> l1.remove(40)

>>> l1

[]

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

>>> l1=[10,20,30,40]

>>> del l1[0:3]

>>> l1

[40]

>>> del l1[0]

>>> l1

[]

>>> del l1

>>> l1-------------NameError: name 'l1' is not defined

>>>

>>>

>>> l1=[10,20,30,40,50,10,20]

>>> print(l1,type(l1),id(l1))

[10, 20, 30, 40, 50, 10, 20] <class 'list'> 2382558920000

>>> del l1[::]

>>> l1

[]

>>> l1=[10,20,30,40,50,10,20]

>>> print(l1,type(l1),id(l1))

[10, 20, 30, 40, 50, 10, 20] <class 'list'> 2382563084288

>>> del l1

>>> l1------------------NameError: name 'l1' is not defined

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

7. copy()

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

=>This Function is used for Copying the content of one object into another object(implement Shallow Copy)

=>Syntax: listobj2=listobj1.copy()

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

Examples: (Shallow Copy)-----Means list objects are same but memory address not

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

>>> l1=[10,"Rossum",23.45]

>>> print(l1,id(l1))

[10, 'Rossum', 23.45] 2382563084288

>>> l2=l1.copy() # Shallow Copy

>>> print(l2,id(l2))

[10, 'Rossum', 23.45] 2382563084416

>>> l1.append("Python")

>>> l2.insert(2,"Django")

>>> print(l1,id(l1))

[10, 'Rossum', 23.45, 'Python'] 2382563084288

>>> print(l2,id(l2))

[10, 'Rossum', 'Django', 23.45] 2382563084416

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

Examples: (Deep Copy)—---list elements and memory address are same on both list.

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

>>> l1=[10,"Rossum",23.45]

>>> print(l1,id(l1))

[10, 'Rossum', 23.45] 2382563084736

>>> l2=l1 # Deep Copy

>>> print(l2,id(l2))

[10, 'Rossum', 23.45] 2382563084736

>>> l1.append("Python")

>>> print(l1,id(l1))

[10, 'Rossum', 23.45, 'Python'] 2382563084736

>>> print(l2,id(l2))

[10, 'Rossum', 23.45, 'Python'] 2382563084736

>>> l2.insert(2,"HYD")

>>> print(l1,id(l1))

[10, 'Rossum', 'HYD', 23.45, 'Python'] 2382563084736

>>> print(l2,id(l2))

[10, 'Rossum', 'HYD', 23.45, 'Python'] 2382563084736

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

Note: SliceBased Copy

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

Examples:

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

>>> l1=[10,20,30,40,50]

>>> print(l1,id(l1))

[10, 20, 30, 40, 50] 2382563084416

>>> l2=l1[::2] # SliceBased Copy

>>> print(l2,id(l2))

[10, 30, 50] 2382563084288

>>> l3=l1[::] #SliceBased Copy

>>> print(l3,id(l3))

[10, 20, 30, 40, 50] 2382563084736

>>> l4=l1[::-1] # SliceBased Copy

>>> print(l4,id(l4))

[50, 40, 30, 20, 10] 2382558906240

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

8. index()

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

=>This function is used for finding the index of First occurrence of specified elements in list object.

=>If the specified element does not exist in list object then we get ValueError

=>Syntax:- varname=lisobj.index(Element)

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

Examples:

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

>>> l1=[10,"Rossum",34.56,True,10]

>>> print(l1)

[10, 'Rossum', 34.56, True, 10]

>>> l1.index("Rossum")

1

>>> l1.index(10)

0

>>> l1.index(True)

3

>>> l1.index("KVR")----------ValueError: 'KVR' is not in list

>>> list().index(10)------------ValueError: 10 is not in list

>>> [].index(0)-----------------ValueError: 0 is not in list

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

9. count()

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

=>This Function is used for counting number of occurrences of Specified element.

=>Syntax: varname=listobj.count(Value)

=>If the specified value does not exist in list object then we get 0

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

Examples:

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

>>> l1=[10,20,30,40,10,10,30,20]

>>> print(l1)

[10, 20, 30, 40, 10, 10, 30, 20]

>>> len(l1)

8

>>> l1.count(10)

3

>>> l1.count(20)

2

>>> l1.count(30)

2

>>> l1.count(40)

1

>>> l1.count(400)

0

>>> list().count(2)

0

>>> [].count(2)

0

>>> l1[::2].count(10)

2

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

10.reverse()

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

=>This Function is used for obtaining Reverse of Original Elements of list.

=>Syntax: listobj.reverse()

Examples:

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

>>> l1=[10,"Rossum","Python",34.56,True,2+3j]

>>> print(l1)----------------[10, 'Rossum', 'Python', 34.56, True, (2+3j)]

>>> l1.reverse()

>>> print(l1)---------------[(2+3j), True, 34.56, 'Python', 'Rossum', 10]

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

11. sort()

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

=>This Function is used for Sorting the given Homogeneous data (similar) in Ascending Order(by default---reverse=False) or Descending Order (reverse=True)

Syntax1: listobj.sort()----->Display the data in Ascending Order [ reverse=False)

(OR)

listobj.sort(reverse=False)

Syntax2: listobj.sort(reverse=True)---Displays the data in Descending Order

Examples:

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

>>> l1=[20,34,12,45,-6,0,23]

>>> print(l1,id(l1))-----------------[20, 34, 12, 45, -6, 0, 23] 2382563092288

>>> l1.sort()

>>> print(l1,id(l1))-------------[-6, 0, 12, 20, 23, 34, 45] 2382563092288

>>> l1.reverse()

>>> print(l1,id(l1))------------------[45, 34, 23, 20, 12, 0, -6] 2382563092288

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

>> l1=[20,34,12,45,-6,0,23]

>>> print(l1,id(l1))

[20, 34, 12, 45, -6, 0, 23] 2382563092224

>>> l1.sort(reverse=True)

>>> print(l1,id(l1))----------------------[45, 34, 23, 20, 12, 0, -6] 2382563092224

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

>>> l1=["Vijay","Bhaskar","Bhasviya","Kaleem","Hari","Trump"]

>>> print(l1,id(l1))---['Vijay', 'Bhaskar', 'Bhasviya', 'Kaleem', 'Hari', 'Trump'] 2382563092288

>>> l1.sort()

>>> print(l1,id(l1))---['Bhaskar', 'Bhasviya', 'Hari', 'Kaleem', 'Trump', 'Vijay'] 2382563092288

>>> l1=["Vijay","Bhaskar","Bhasviya","Kaleem","Hari","Trump"]

>>> print(l1,id(l1))----['Vijay', 'Bhaskar', 'Bhasviya', 'Kaleem', 'Hari', 'Trump'] 2382563092224

>>> l1.sort(reverse=True)

>>> print(l1,id(l1))-----['Vijay', 'Trump', 'Kaleem', 'Hari', 'Bhasviya', 'Bhaskar'] 2382563092224

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

>>> l1=[10,"Trump",True,2+3.5j]

>>> print(l1,id(l1))

[10, 'Trump', True, (2+3.5j)] 2382563092288

>>> l1.sort()---------------TypeError: '<' not supported between instances of 'str' and 'int'

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

>>> l1=[20,34,12,45,-6,0,23]

>>> print(l1,id(l1))-----------------------[20, 34, 12, 45, -6, 0, 23] 2382563092224

>>> l1.sort(reverse=False)

>>> print(l1,id(l1))-------------------[-6, 0, 12, 20, 23, 34, 45] 2382563092224

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

>>> l1=[20,34,12,45,-6,0,23]

>>> print(l1,id(l1))

[20, 34, 12, 45, -6, 0, 23] 2382563092288

>>> l1.sort()

>>> print(l1,id(l1))----------------[-6, 0, 12, 20, 23, 34, 45] 2382563092288

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

12. extend()

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

=>This function is used for adding the destination list object to source list object

=>Syntax: sourcelistobj.extend(destination object)

Examples:

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

>>> l1=[10,20,30,40]

>>> l2=["python","Django","Data Sci","Java"]

>>> l1.extend(l2)

>>> print(l1)----------------[10, 20, 30, 40, 'python', 'Django', 'Data Sci', 'Java']

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

>>> l1=[10,20,30]

>>> l2=[1.2,1.3,1.4]

>>> l3=["Python","Data Sci"]

>>> l1.extend(l2,l3)------------TypeError: list.extend() takes exactly one argument (2 given)

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

We can use + operator for extending multiple list objects content into single list

Syntax:

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

listobj-n= listobj1+listobj2+........listobj-n

>>> l1=[10,20,30]

>>> l2=[1.2,1.3,1.4]

>>> l3=["Python","Data Sci"]

>>> l1.extend(l2,l3)------------TypeError: list.extend() takes exactly one argument (2 given)

>>> print(l1)-------------[10, 20, 30]

>>> print(l2)------------[1.2, 1.3, 1.4]

>>> print(l3)------------['Python', 'Data Sci']

>>> l1=l1+l2+l3

>>> print(l1)----------[10, 20, 30, 1.2, 1.3, 1.4, 'Python', 'Data Sci']

>>> print(l2)----------[1.2, 1.3, 1.4]

>>> print(l3)---------['Python', 'Data Sci']

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

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

Inner or Nested List

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

=>The process of defining one list inside of another list is called Inner or Nested List.

=>Syntax:

listobj=[ Val1,Val2.....[val11,val12,...val1n],....[Val21,Val22...Val2n].....]

=>Here "listobj" is called Outer List

=>Here [val11,val12,...val1n] is called Inner List of Outer List

=>Here [Val21,Val22...Val2n] is also another inner list of Outer List

=>On the Inner List , we can also apply all pre-defined Functions of List.

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

Examples:

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

>>> studlist=[ 10,"Naveen", [18,19,17] , [78,66,77], "OUCET" ]

>>> print(studlist)

[10, 'Naveen', [18, 19, 17], [78, 66, 77], 'OUCET']

>>> print(studlist,type(studlist))

[10, 'Naveen', [18, 19, 17], [78, 66, 77], 'OUCET'] <class 'list'>

>>> studlist[0]

10

>>> studlist[1]

'Naveen'

>>> studlist[2]

[18, 19, 17]

>>> studlist[3]

[78, 66, 77]

>>> studlist[-1]

'OUCET'

>>> studlist[-2]

[78, 66, 77]

>>> studlist[-3]

[18, 19, 17]

>>> print(studlist,type(studlist))

[10, 'Naveen', [18, 19, 17], [78, 66, 77], 'OUCET'] <class 'list'>

>>> studlist[2:4]

[[18, 19, 17], [78, 66, 77]]

>>> studlist.append("HYD")

>>> print(studlist,type(studlist))

[10, 'Naveen', [18, 19, 17], [78, 66, 77], 'OUCET', 'HYD'] <class 'list'>

>>> studlist[2].append(16)

>>> print(studlist,type(studlist))

[10, 'Naveen', [18, 19, 17, 16], [78, 66, 77], 'OUCET', 'HYD'] <class 'list'>

>>> studlist[-3].insert(1,54)

>>> print(studlist,type(studlist))

[10, 'Naveen', [18, 19, 17, 16], [78, 54, 66, 77], 'OUCET', 'HYD'] <class 'list'>

>>> studlist[2].sort(reverse=False)

>>> print(studlist,type(studlist))

[10, 'Naveen', [16, 17, 18, 19], [78, 54, 66, 77], 'OUCET', 'HYD'] <class 'list'>

>>> studlist[-3].sort(reverse=True)

>>> print(studlist,type(studlist))

[10, 'Naveen', [16, 17, 18, 19], [78, 77, 66, 54], 'OUCET', 'HYD'] <class 'list'>

>>> studlist[2].remove(17)

>>> print(studlist,type(studlist))

[10, 'Naveen', [16, 18, 19], [78, 77, 66, 54], 'OUCET', 'HYD'] <class 'list'>

>>> studlist[-3].pop(-2)

66

>>> print(studlist,type(studlist))

[10, 'Naveen', [16, 18, 19], [78, 77, 54], 'OUCET', 'HYD'] <class 'list'>

>>> del studlist[2]

>>> print(studlist,type(studlist))

[10, 'Naveen', [78, 77, 54], 'OUCET', 'HYD'] <class 'list'>

>>> studlist.pop(2)

[78, 77, 54]

>>> print(studlist,type(studlist))

[10, 'Naveen', 'OUCET', 'HYD'] <class 'list'>

>>> studlist.insert(2,[15,16,18,17])

>>> print(studlist,type(studlist))

[10, 'Naveen', [15, 16, 18, 17], 'OUCET', 'HYD'] <class 'list'>

>>> studlist.append([65,66,78,57])

>>> print(studlist,type(studlist))

[10, 'Naveen', [15, 16, 18, 17], 'OUCET', 'HYD', [65, 66, 78, 57]] <class 'list'>

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

My Requirement

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

I want to store stno,name, Internal Marks, External Marks, collname

stud= [ 10 "Naveen", 18,19,17, 78,66,77, "OUCET" ]--No recommended

(OR)

studlist= [ 10 "Naveen", [18,19,17] , [78,66,77], "OUCET" ]--Recommended

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

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

tuple

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

Properties of tuple:

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

=>’tuple' is one of the pre-defined class and treated as list data type.

=>The purpose of tuple data type is that "To store multiple values either of Same

Type or Different Type or Both the types in single Object with Unique and Duplicates."

=>The elements of tuple must be enclosed within Braces( ) and elements must

separated by comma.

=>An object of tuple maintains Insertion Order. i.e., In whichever Order, we insert the data, in the same order we get the output.

=>On the object tuple we can perform Both Indexing and Slicing Operations.

=>An object of tuple belongs to immutable.

=>We can convert any type of value into tuple type of values by using tuple()

Syntax: tplobj=tuple(object)

=>We can create two types of tuple objects. They are

1) empty tuple

2) non-empty tuple

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

=>Empty tuple

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

=>An empty tuple is one, which does not contain any elements and whose length=0

=>Syntax:- tplobj=( )

(OR

tplobj=tuple()

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

=>Non-Empty tuple

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

=>A non-empty tuple is one, which contains elements and whose length>0

=>Syntax:- tplobj=(Val1,Val2.....Val-n )

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

Note: The Functionality of Tuple is Exactly like List. But An object of tuple belongs to Immutable and object list belongs mutable.

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

Examples:

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

>>> t1=(10,20,30,40,10,15)

>>> print(t1,type(t1))

(10, 20, 30, 40, 10, 15) <class 'tuple'>

>>> t1=(10,"abinash","Python",45.67)

>>> print(t1,type(t1))

(10, 'abinash', 'Python', 45.67) <class 'tuple'>

>>>

>>> print(t1)

(10, 'abinash', 'Python', 45.67)

>>> t1[0]

10

>>> t1[0:2]

(10, 'abinash')

>>> t1[::2]

(10, 'Python')

>>> t1[::-1]

(45.67, 'Python', 'abinash', 10)

>>>

>>> print(t1,id(t1))

(10, 'abinash', 'Python', 45.67) 2302868231952

>>> t1[2]="Data Sci"

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: 'tuple' object does not support item assignment

>>>

>>>

>>> l1=[10,20,-24,-56,"Hyd"]

>>> print(l1,type(l1))

[10, 20, -24, -56, 'Hyd'] <class 'list'>

>>> t2=tuple(l1)

>>> print(t2,type(t2))

(10, 20, -24, -56, 'Hyd') <class 'tuple'>

>>>

>>> t1=(10,"abinash","Python",45.67)

>>> print(t1,type(t1))

(10, 'abinash', 'Python', 45.67) <class 'tuple'>

>>> len(t1)

4

>>> t2=()

>>> print(t2,type(t2))

() <class 'tuple'>

>>> len(t2)

0

>>> t3=tuple()

>>> print(t3,type(t3))

() <class 'tuple'>

>>> len(t3)

0

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

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

Pre-defined Functions in tuple

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

=>The Pre-defined Functions in tuple are

1) index()

2) count()

Examples:

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

>>> t1=(10,20)

>>> print(t1,type(t1))-----------(10, 20) <class 'tuple'>

>>> t1.index(10)----------------0

>>> t1.index(20)----------------1

>>> t1.index(200)------------ValueError: tuple.index(x): x not in tuple

>>> t1=(10,10,10,20,20,30,50)

>>> t1.count(10)----------------3

>>> t1.count(20)--------------2

>>> t1.count(30)-------------1

>>> t1.count(430)--------------0

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

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

Inner or Nested Tuple

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

=>The process of defining one tuple inside of another tuple is called Inner or Nested tuple.

=>Syntax:-

tplobj=( Val1,Val2.....(val11,val12,...val1n),....(Val21,Val22...Val2n).....)

=>Here "tplobj" is called Outer tuple

=>Here (val11,val12,...val1n) is called Inner tuple of Outer Tuple

=>Here (Val21,Val22...Val2n) is also another inner tuple of Outer tuple

=>On the Inner tuple , we can also apply all pre-defined Functions of tuple.

Note:- We can write One list in another list.

We can write one tuple in another tuple.

we can write one list in tuple.

We can write one tuple in list.

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

Examples:-

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

>>> t1=(10,"Travis",("NumLib1.2","Numpy1.22"),["RS","PYTHON"], "NL")

>>> print(t1,type(t1))

(10, 'Travis', ('NumLib1.2', 'Numpy1.22'), ['RS', 'PYTHON'], 'NL') <class 'tuple'>

>>> print(t1[2],t1[3])

('NumLib1.2', 'Numpy1.22') ['RS', 'PYTHON']

>>> print(t1[2],type(t1[2], t1[3]),type(t1[3]))

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: type() takes 1 or 3 arguments

>>> print(t1[2],type(t1[2]), t1[3],type(t1[3]))

('NumLib1.2', 'Numpy1.22') <class 'tuple'> ['RS', 'PYTHON'] <class 'list'>

>>> t1[2].append(20)

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

AttributeError: 'tuple' object has no attribute 'append'

>>> t1[3].append("R")

>>> print(t1)

(10, 'Travis', ('NumLib1.2', 'Numpy1.22'), ['RS', 'PYTHON', 'R'], 'NL')

>>>

>>>

>>> l1=[10,"Travis",(10,20),[30,40],"HYD"]

>>> print(l1,type(l1))

[10, 'Travis', (10, 20), [30, 40], 'HYD'] <class 'list'>

>>> print(l1[2],type(l1[2]))

(10, 20) <class 'tuple'>

>>> print(l1[3],type(l1[3]))

[30, 40] <class 'list'>

>>> l1[2].insert(1,20)---------------AttributeError: 'tuple' object has no attribute 'insert'

>>> l1[3].insert(1,20)

>>> print(l1,type(l1))

[10, 'Travis', (10, 20), [30, 20, 40], 'HYD'] <class 'list'>

>>>

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

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

Types of Copy Techniques

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

=>In Python Programming, we have two types of Copy Techniques. They are

1. Shallow Copy

2. Deep Copy

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

1. Shallow Copy

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

=>The Shallow Copy Satisfies the following Properties.

a) Initial Content of Both the objects are Same.

b) The Memory Address of Both the objects are Different.

c) The modifications / updating is Independent ( i.e., Modification is not reflected to each other).

=>To Implement Shallow Copy , we use Copy()

=>Syntax: desorb=srcobj.copy()

Examples:

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

>>> l1=[10,"Rossum",23.45]

>>> print(l1,id(l1))

[10, 'Rossum', 23.45] 2382563084288

>>> l2=l1.copy() # Shallow Copy

>>> print(l2,id(l2))

[10, 'Rossum', 23.45] 2382563084416

>>> l1.append("Python")

>>> l2.insert(2,"Django")

>>> print(l1,id(l1))

[10, 'Rossum', 23.45, 'Python'] 2382563084288

>>> print(l2,id(l2))

[10, 'Rossum', 'Django', 23.45] 2382563084416

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

2. Deep Copy

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

=>The Deep Copy Satisfies the following Properties.

a) Initial Content of Both the objects are Same.

b) The Memory Address of Both the objects are Same.

c) The modifications / updating is Dependent ( i.e., Modifications are reflected to each other).

=>To Implement Deep Copy, we use assignment Operator ( = )

=>Syntax:- destobj=srcobj

Examples:

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

>>> l1=[10,"Rossum",23.45]

>>> print(l1,id(l1))

[10, 'Rossum', 23.45] 2382563084736

>>> l2=l1 # Deep Copy

>>> print(l2,id(l2))

[10, 'Rossum', 23.45] 2382563084736

>>> l1.append("Python")

>>> print(l1,id(l1))

[10, 'Rossum', 23.45, 'Python'] 2382563084736

>>> print(l2,id(l2))

[10, 'Rossum', 23.45, 'Python'] 2382563084736

>>> l2.insert(2,"HYD")

>>> print(l1,id(l1))

[10, 'Rossum', 'HYD', 23.45, 'Python'] 2382563084736

>>> print(l2,id(l2))

[10, 'Rossum', 'HYD', 23.45, 'Python'] 2382563084736

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

Note: SliceBased Copy

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

Examples:

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

>>> l1=[10,20,30,40,50]

>>> print(l1,id(l1))

[10, 20, 30, 40, 50] 2382563084416

>>> l2=l1[::2] # SliceBased Copy

>>> print(l2,id(l2))

[10, 30, 50] 2382563084288

>>> l3=l1[::] #SliceBased Copy

>>> print(l3,id(l3))

[10, 20, 30, 40, 50] 2382563084736

>>> l4=l1[::-1] # SliceBased Copy

>>> print(l4,id(l4))

[50, 40, 30, 20, 10] 2382558906240

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

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

Set Category data Types (Collections Data Types or Data Structures)

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

The purpose of Set Category data Types is that "To store multiple values either of Same Type or Different Type or Both the types in single Object with Unique Values only (Duplicates are not allowed.)"

=>Set Category contains two data types They are

1) set (mutable and Immutable)

2) frozenset (immutable)

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

1) set (mutable and Immutable)

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

=>'set' is one the pre-defined class and treated as set data types.

=>The purpose of set data type is that " To store multiple values either of Same Type or Different Type or Both the types in single Object with Unique Values only (Duplicates are not allowed.)"

=>The elements of set must be enclosed within curly braces { } and its values separated by comma.

=>An object of set does not main insertion Order because It can display any one of the possibility of set elements.

=>On the object of set , we cannot perform Indexing and Slicing Operations because set object does not main insertion Order.

=>An object of set belongs to Mutable in the case add() and immutable in the case Item Assignment ( set' object does not support item assignment).

=>We have two types of sets. They are

a) Empty Set

b) Non-Empty Set

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

a) Empty Set

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

=>An empty set is one, which does not contain any elements and whose length=0.

=>Syntax:- setobj= set()

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

b) Non-Empty Set

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

=>A Non-Empty set is one, which contains elements and whose length>0.

=>Syntax:- setobj= {Val1,Val2,,------Val-n}

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

Examples:

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

>>> s1={10,20,30,40,50,60,10,20}

>>> print(s1,type(s1))---------------------{50, 20, 40, 10, 60, 30} <class 'set'>

>>> s2={10,"Vijay",34.56,"python",True}

>>> print(s2,type(s2))---------------------{True, 34.56, 'Vijay', 10, 'python'} <class 'set'>

>>> s2[0]----------------------------TypeError: 'set' object is not subscriptable

>>> s2[2:4]--------------------------TypeError: 'set' object is not subscriptable

>>> s2={10,"Vijay",34.56,"python",True}

>>> s2[0]=100------------------------TypeError: 'set' object does not support item assignment

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

>>> s2={10,"Vijay",34.56,"python",True}

>>> print(s2,type(s2),id(s2))-------{True, 34.56, 'Vijay', 10, 'python'} <class 'set'> 1634682621280

>>> s2.add("Django")

>>> print(s2,type(s2),id(s2))---{True, 34.56, 'Vijay', 10, 'Django', 'python'} <class 'set'> 1634682621280

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

>>> s={}

>>> print(s,type(s))-----------------{} <class 'dict'>

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

>>> s1=set()

>>> print(s1,type(s1))-----------------set() <class 'set'>

>>> len(s1)-----------------------0

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

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

Pre-defined Functions in set

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

=>set object contains different types of Pre-defined Functions which are performing different types of Operations.

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

1) add()

=>This Function is uaed for adding an element to the set object

=>Syntax:- setobj.add(value)

Examples:

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

>>> s1={10,"Hari"}

>>> print(s1,type(s1),id(s1))---------------{'Hari', 10} <class 'set'> 1634678630976

>>> s1.add("NagarjunN")

>>> print(s1,type(s1),id(s1))---------{'Hari', 10, 'NagarjunN'} <class 'set'> 1634678630976

>>> s1.add(23.45)

>>> print(s1,type(s1),id(s1))--------{'Hari', 10, 23.45, 'NagarjunN'} <class 'set'> 1634678630976

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

2) remove()

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

=>This function is used for removing the value(key) from set object.

=>Syntax: setobj.remove(Value)

=>If the value does not exist in set object then we get KeyError.

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

Examples:

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

>>> s1={10,"Kumar","Python",34.56,"Java",True}

>>> print(s1,type(s1),id(s1))

{True, 34.56, 'Python', 'Java', 10, 'Kumar'} <class 'set'> 1634682621728

>>> s1.remove("Python")

>>> print(s1,type(s1),id(s1))

{True, 34.56, 'Java', 10, 'Kumar'} <class 'set'> 1634682621728

>>> s1.remove("Java")

>>> print(s1,type(s1),id(s1))

{True, 34.56, 10, 'Kumar'} <class 'set'> 1634682621728

>>> s1.remove(10)

>>> print(s1,type(s1),id(s1))

{True, 34.56, 'Kumar'} <class 'set'> 1634682621728

>>> s1.remove(100)----------KeyError: 100

>>> set().remove(23)-----------KeyError: 23

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

3) discard()

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

=>This Function is used for removing the value from setobj

=>Syntax: setobj.discard(value)

=>if the value does not exist in set object then we never get any error.

NOTE:- remove() generates KeyError if the value does not exist

discard() never generates KeyError if the value does not exist

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

Examples:

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

>>> s1={10,"Kumar","Python",34.56,"Java",True}

>>> print(s1,type(s1),id(s1))

{True, 34.56, 'Python', 'Java', 10, 'Kumar'} <class 'set'> 1634678630976

>>> s1.discard("Java")

>>> print(s1,type(s1),id(s1))

{True, 34.56, 'Python', 10, 'Kumar'} <class 'set'> 1634678630976

>>> s1.discard(34.56)

>>> print(s1,type(s1),id(s1))

{True, 'Python', 10, 'Kumar'} <class 'set'> 1634678630976

>>> s1.discard("Django")

>>> print(s1,type(s1),id(s1))

{True, 'Python', 10, 'Kumar'} <class 'set'> 1634678630976

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

4) pop()

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

=>This Function is used for removing Arbitary Element/ Value / Key from Set Object

=>Syntax:- setobj.pop()

=>When we call pop() upon empty set object then we get KeyError.

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

Examples:

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

>>> s1={10,"Kumar","Python",34.56,"Java",True}

>>> s1.pop()

True

>>> s1.pop()

34.56

>>> s1.pop()

'Python'

>>> s1.pop()

'Java'

>>> s1.pop()

10

>>> s1.pop()

'Kumar'

>>> print(s1,type(s1),id(s1))

set() <class 'set'> 1634678630976

>>> s1.pop()----------------KeyError: 'pop from an empty set'

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

>>> s1={10,"Kumar","Python",34.56,"Java",True}

>>> s1.pop()

True

>>> s1.pop()

34.56

>>> s1.pop()

'Python'

>>> s1.pop()

'Java'

>>> s1.pop()

10

>>> s1.pop()

'Kumar'

>>> print(s1,type(s1),id(s1))

set() <class 'set'> 1634678630976

>>> s1.pop()-----------------KeyError: 'pop from an empty set'

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

>>> s1={10,20,30,40,50,60,70,80,10}

>>> print(s1,type(s1),id(s1))

{70, 40, 10, 80, 50, 20, 60, 30} <class 'set'> 1634682621728

>>> s1.pop()

70

>>> s1.pop()

40

>>> s1.pop()

10

>>> s1.pop()

80

>>> s1.pop()

50

>>> s1.pop()

20

>>> s1.pop()

60

>>> s1.pop()

30

>>> s1.pop()-------------------KeyError: 'pop from an empty set'

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

>>> s1={"Kaleem","Bharani","Dynaneswar","Rossum","Travis","Ram"}

>>> s1.pop()

'Ram'

>>> s1.pop()

'Kaleem'

>>> print(s1,type(s1),id(s1))

{'Travis', 'Bharani', 'Rossum', 'Dynaneswar'} <class 'set'> 1634678630976

>>> s1.pop()

'Travis'

>>> s1.pop()

'Bharani'

>>> s1.pop()

'Rossum'

>>> s1.pop()

'Dynaneswar'

>>> s1.pop()----------------KeyError: 'pop from an empty set'

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

5) copy():

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

=>This Function is used for copying the content of One setobject into another setobject ( Implementation of shallow Copy)

=>Syntax: setobj2=setobj1.copy()

Examples:(shallow copy)

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

>>> s1={10,"rambabu"}

>>> print(s1,id(s1))

{10, 'rambabu'} 1634682621728

>>> s2=s1.copy()

>>> print(s2,id(s2))

{10, 'rambabu'} 1634678630976

>>> s1.add("Python")

>>> s2.add("Django")

>>> print(s1,id(s1))

{10, 'rambabu', 'Python'} 1634682621728

>>> print(s2,id(s2))

{10, 'rambabu', 'Django'} 1634678630976

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

Examples:(Deep copy)

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

>>> s1={10,"rambabu"}

>>> print(s1,id(s1))

{10, 'rambabu'} 1634682621280

>>> s2=s1

>>> print(s2,id(s2))

{10, 'rambabu'} 1634682621280

>>>

>>>

>>> s1.add("Python")

>>> print(s1,id(s1))

{10, 'rambabu', 'Python'} 1634682621280

>>> print(s2,id(s2))

{10, 'rambabu', 'Python'} 1634682621280

NOTE:

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

>>> s1=set()

>>> print(s1,id(s1))-----------set() 1634678630976

>>> s2=s1.copy()

>>> print(s2,id(s2))-------------set() 1634682621728

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

6) isdisjoint()

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

=>Syntax: setobj1.isdisjoint(setobj2)

=>This Function returns True provided there is no common element in setobj1 and setobj2.

=>This Function returns False provided there is common element in setobj1 and setobj2.

Examples:

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

>>> s1={10,20,30,40}

>>> s2={15,25,35}

>>> s3={10,45,55,65}

>>> s1.isdisjoint(s2)-----------True

>>> s1.isdisjoint(s3)----------False

>>> s2.isdisjoint(s3)------------True

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

7) issubset()

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

=>Syntax:- setobj1.issubset(setobj2)

=>This Function returns True provided all the elements of setobj1 are present in setobj2

otherwise, it returns False

Examples:

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

>>> s1={10,20,30,40}

>>> s2={10,20}

>>> s3={15,25,35,45}

>>> s4={10,70,80}

>>> s2.issubset(s1)------------------True

>>> s3.issubset(s1)-----------------False

>>> s4.issubset(s1)-----------------False

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

8) issuperset()

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

=>Syntax:- setobj1.issuperset(setobj2)

=>This Function returns True provided all the elements of setobj2 are present in setobj1

otherwise, it returns False

Examples:

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

>>> s1={10,20,30,40}

>>> s2={10,20}

>>> s3={20,40,50,60}

>>> s1.issuperset(s2)-------------True

>>> s1.issuperset(s3)------------False

>>> s1={"Sachin","Virat","Rohit","Pandey"}

>>> s2={"Sachin","Virat","Rohit","Pandey"}

>>> s1.issubset(s2)-----------True

>>> s1.issuperset(s2)--------True

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

9) union()

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

=>Syntax: setobj3=setobj1.union(setobj2)

(OR)

setobj1.union(setobj2)

=>This function is used for takling all unique elements of setobj1 and setobj2

Examples:

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

>>> s1={10,20,30,40}

>>> s2={10,20,50,60}

>>> s3=s1.union(s2)

>>> print(s3,type(s3))----------------{40, 10, 50, 20, 60, 30} <class 'set'>

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

10) intersection()

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

=>Syntax: setobj3=setobj1.intersection(setobj2)

(OR)

setobj1.intersection(setobj2)

=>This Function is used for obtaining common elements from setobj1 and setobj2

Examples:

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

>>> s1={10,20,30,40}

>>> s2={10,20,50,60}

>>> s3=s1.intersection(s2)

>>> print(s3,type(s3))-------------{10, 20} <class 'set'>

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

11) difference()

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

=>Syntax: setobj3=setobj1.difference(setobj2)

=>This Function Elements / removes common elements from setobj1 and setobj2 and takes

remaining elements from setobj1 and place them setobj3

Examples:

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

>>> s1={10,20,30,40}

>>> s2={10,20,50,60}

>>> s3=s1.difference(s2)

>>> print(s3,type(s3))----------------{40, 30} <class 'set'>

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

=>Syntax:- setobj3=setobj2.difference(setobj1)

=>This Function Eliminates/ removes common elements from setobj2 and setobj1 and takes

remaining elements from setobj2and place them setobj3

Examples:

>>> s1={10,20,30,40}

>>> s2={10,20,50,60}

>>> s3=s2.difference(s1)

>>> print(s3,type(s3))---------------{50, 60} <class 'set'>

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

12) symmetric difference()

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

=>Syntax:- setobj3=setobj2.symmetric\_difference()(setobj1)

=>This Function Eliminates/ removes common elements from setobj2 and setobj1 and takes

remaining elements from setobj2 and setobj1 and place them setobj3

Examples:

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

>>> s1={10,20,30,40}

>>> s2={10,20,50,60}

>>> s3=s1.symmetric\_difference(s2)

>>> print(s3,type(s3))--------------{40, 50, 60, 30} <class 'set'>

(OR)

>>> s3=s1.union(s2).difference(s1.intersection(s2) )

>>> print(s3,type(s3))------------{40, 50, 60, 30} <class 'set'>

Note:

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

>>> s1={10,20,30,40}

>>> s2={10,20,50,60}

>>> s3={'a','b'}

>>> s4=s1.union(s2,s3)

>>> print(s4)---------------{'b', 40, 10, 50, 20, 'a', 60, 30}

>>> s4=s1.union(s2).union(s3)

>>> print(s4)----------------{'b', 50, 20, 'a', 40, 10, 60, 30}

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

13) update()

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

Syntax: setobj1.update(setobj2)

=>This function is used for updating the elements of setobj1 with setobj2.

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

Examples:

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

>>> s1={10,20,30,40}

>>> s2={35,45}

>>> s1.update(s2)

>>> print(s1,type(s1))-----------{35, 20, 40, 10, 45, 30} <class 'set'>

>>> s3={10,20,30,40}

>>> s1.update(s3)

>>> print(s1)---------------------{35, 40, 10, 45, 20, 30}

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

Use Examples:

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

>>> cp={"sachin","rohit","virat","pandey"}

>>> tp={"rossum","sachin","dennis"}

>>> allcptp=cp.union(tp)

>>> print(allcptp)-------{'pandey', 'virat', 'sachin', 'dennis', 'rohit', 'rossum'}

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

>>> bothcptp=cp.intersection(tp)

>>> print(bothcptp)----------------------{'sachin'}

(OR)

>>> bothcptp=tp.intersection(cp)

>>> print(bothcptp)----------------{'sachin'}

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

>>> onlycp=cp.difference(tp)

>>> print(onlycp)-----------{'pandey', 'rohit', 'virat'}

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

>>> onlytp=tp.difference(cp)

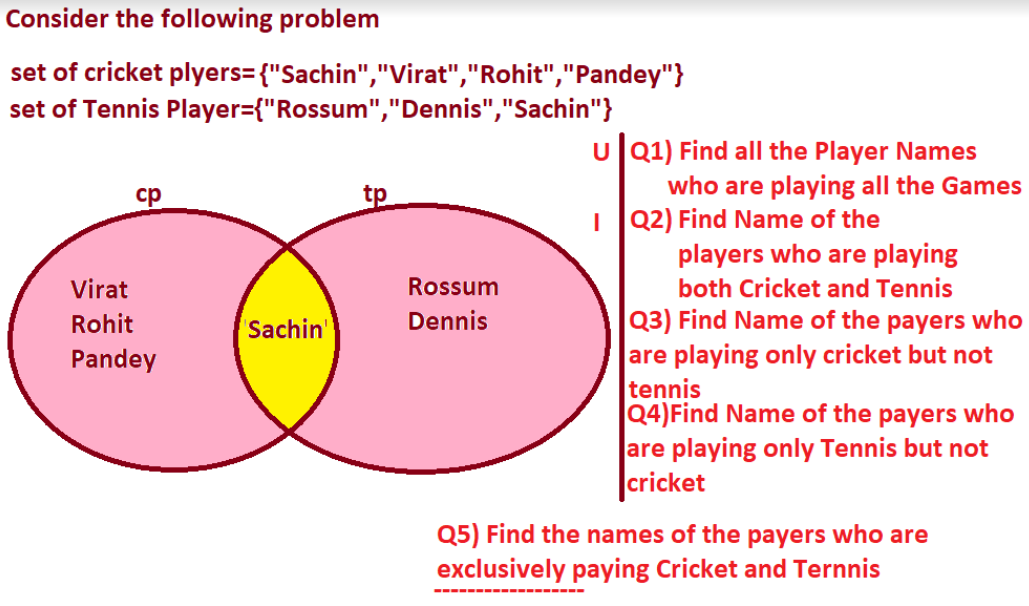
>>> print(onlytp)--------------{'dennis', 'rossum'}

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

>>> exclcptp=cp.symmetric\_difference(tp)

>>> print(exclcptp)---------------{'pandey', 'virat', 'dennis', 'rohit', 'rossum'}

===============================X======================================



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

frozenset

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

=>'frozenset' is one the pre-defined class and treated as set data types.

=>The purpose of frozenset data type is that " To store multiple values either of Same Type or

Different Type or Both the types in single Object with Unique Values only (Duplicates are not allowed.)"

=>There is no symbolic Notation for representing the elements of frozenset but we can convert other type of elements into frozenset by using frozenset()

Syntax:

frozenness=frozenset(object)

=>An object of frozenset does not main insertion Order because It can display any one of the possibility of frozenset elements.

=>On the object of frozenset, we cannot perform Indexing and Slicing Operations because set object does not main insertion Order.

=>An object of frozenset belongs to immutable because neither allows item assignment nor allows add()

----------

Note:

---------

=>The Functionality of frozenset is exactly like set. An object frozenset belongs to immutable (neither allows item assignment nor allows add() ) where as an object set belongs to both Mutable ( in the case of add()) and immutable in the case of item assignment.

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

>>> s1={10,20,30,40,50,60,10}

>>> print(s1,type(s1))----------------{50, 20, 40, 10, 60, 30} <class 'set'>

>>> fs=frozenset(s1)

>>> print(fs,type(fs))-----------------frozenset({50, 20, 40, 10, 60, 30}) <class 'frozenset'>

>>> l1=[10,"Vivek",45.66,"Python"]

>>> print(l1,type(l1))-----------------[10, 'Vivek', 45.66, 'Python'] <class 'list'>

>>> fs1=frozenset(l1)

>>> print(fs1,type(fs1),id(fs1))----------frozenset({'Vivek', 10, 'Python', 45.66}) <class 'frozenset'> 2400159812960

>>> fs1[0]-----------------TypeError: 'frozenset' object is not subscriptable

>>> fs1[0:3]--------------TypeError: 'frozenset' object is not subscriptable

>>> fs1[0]=345------------TypeError: 'frozenset' object does not support item assignment

>>> fs1.add(100)---------AttributeError: 'frozenset' object has no attribute 'add'

=============================X===================================================

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

pre-defined Functions in frozenset

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

=>On The frozenset object, we can perform some additional Operations by using the following Functions,

copy()

isdisjoint()

issuperset()

issubset()

union()

intersection()

difference()

symmetric\_difference()

=>frozenset object does not contain the following pre-defined functions, because frozenset object is immutable.

add()

remove()

pop()

discard()

update()

=============================X===================================================

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

V) Dict Category Data Types ( Collection Data Types or Data Structures)

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

=>'dict' is one of the pre-defined class and treated as Dict Data Type.

=>The purpose of dict data type is that "To store (Key,value) in single variable"

=>In (Key, Value), the value of Key is Unique and Value of Value may or may not be unique.

=>The (Key, value) must be organized or stored in the object of dict within Curly Braces {} and they separated by comma.

=>An object of dict does not support Indexing and Slicing bcoz Values of Key itself considered as Indices.

=>In the object of dict, Values of Key are treated as Immutable and Values of Value are treated as mutable.

=>Originally an object of dict is mutable bcoz we can add (Key,Value) externally.

=>We have two types of dict objects. They are

a) Empty dict

b) Non-empty dict

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

a) Empty dict

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

=>Empty dict is one, which does not contain any (Key,Value) and whose length is 0

=>Syntax:- dictobj1= { }

or

dictobj=dict()

=>Syntax for adding (Key,Value) to empty dict:

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

dictobj[Key1]=Val1

dictobj[Key2]=Val2

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

dictobj[Key-n]=Val-n

Here Key1,Key2...Key-n are called Values of Key and They must be Unique

Here Val1, Val2...Val-n are called Values of Value and They may or may not be unique.

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

b) Non-Empty dict

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

=>Non-Empty dict is one, which contains (Key,Value) and whose length is >0

=>Syntax:- dictobj1= { Key1:Val1,Key2:Val2......Key-n:Valn}

Here Key1,Key2...Key-n are called Values of Key and They must Unique

Here Val1, Val2...Val-n are called Values of Value and They may or may not be unique.

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

Examples:

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

>>> d1={10:"Python",20:"Data Sci",30:"Django"}

>>> print(d1,type(d1))----------{10: 'Python', 20: 'Data Sci', 30: 'Django'} <class 'dict'>

>>> d2={10:3.4,20:4.5,30:5.6,40:3.4}

>>> print(d2,type(d2))------------{10: 3.4, 20: 4.5, 30: 5.6, 40: 3.4} <class 'dict'>

>>> len(d1)--------------3

>>> len(d2)------------4

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

>>> d3={}

>>> print(d3,type(d3))------------{} <class 'dict'>

>>> len(d3)-------------0

>>> d4=dict()

>>> print(d4,type(d4))-------------{} <class 'dict'>

>>> len(d4)---------------0

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

>>> d2={10:3.4,20:4.5,30:5.6,40:3.4}

>>> print(d2)----------------------------------{10: 3.4, 20: 4.5, 30: 5.6, 40: 3.4}

>>> d2[0]----------------------------------------KeyError: 0

>>> d2[10]-------------------------------------3.4

>>> d2[10]=10.44

>>> print(d2)------------------------------{10: 10.44, 20: 4.5, 30: 5.6, 40: 3.4}

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

>>> d2={10:3.4,20:4.5,30:5.6,40:3.4}

>>> print(d2,type(d2),id(d2))----{10: 3.4, 20: 4.5, 30: 5.6, 40: 3.4} <class 'dict'> 2090750380736

>>> d2[50]=5.5

>>> print(d2,type(d2),id(d2))---{10: 3.4, 20: 4.5, 30: 5.6, 40: 3.4, 50: 5.5} <class 'dict'> 2090750380736

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

>>> d3={}

>>> print(d3,type(d3),id(d3))-------------------{} <class 'dict'> 2090750332992

>>> d3["Python"]=1

>>> d3["Java"]=3

>>> d3["C"]=2

>>> d3["GO"]=1

>>> print(d3,type(d3),id(d3))-----{'Python': 1, 'Java': 3, 'C': 2, 'GO': 1} <class 'dict'> 2090750332992

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

>>> d4=dict()

>>> print(d4,type(d4),id(d4))---------------{} <class 'dict'> 2090754532032

>>> d4[10]="Apple"

>>> d4[20]="Mango"

>>> d4[30]="Kiwi"

>>> d4[40]="Sberry"

>>> d4[50]="Orange"

>>> print(d4,type(d4),id(d4))---{10: 'Apple', 20: 'Mango', 30: 'Kiwi', 40: 'Sberry', 50: 'Orange'}<class 'dict'> 2090754532032

>>> d4[10]="Guava"

>>> print(d4,type(d4),id(d4))----{10: 'Guava', 20: 'Mango', 30: 'Kiwi', 40: 'Sberry', 50: 'Orange'} <class 'dict'>2090754532032

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

>>> d2={10:3.4,20:4.5,30:5.6,40:3.4}

>>> print(d2,type(d2),id(d2))---{10: 3.4, 20: 4.5, 30: 5.6, 40: 3.4}

<class 'dict'> 2090754531520

>>> d2[50]=1.2

>>> print(d2,type(d2),id(d2))---{10: 3.4, 20: 4.5, 30: 5.6, 40: 3.4, 50: 1.2}

<class 'dict'> 2090754531520

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

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

Pre-defined functions in dict

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

=>On the object of dict, we can perform the following operations by using pre-defined functions present in dict object.

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

1) clear()

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

=>This function is used for removing all the elements of dict object.

=>Syntax: dictobj.clear()

Example:

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

>>> d1={10:"Apple",20:"Mango",30:"Kiwi",40:"Banana"}

>>> print(d1,type(d1),id(d1))----{10: 'Apple', 20: 'Mango', 30: 'Kiwi', 40: 'Banana'} <class 'dict'> 2400155649792

>>> len(d1)-------4

>>> d1.clear()

>>> print(d1,type(d1),id(d1))-----------{} <class 'dict'> 2400155649792

>>> len(d1)-------------0

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

2) pop()

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

=>This function is used for removing the (Key,Value) from dict object by passing value of Key.

=>Syntax:- dictobj.pop(key)

=>If the key does not exist then we get KeyError

Examples:

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

>>> d1={10:"Apple",20:"Mango",30:"Kiwi",40:"Banana"}

>>> print(d1,type(d1),id(d1))------{10: 'Apple', 20: 'Mango', 30: 'Kiwi', 40: 'Banana'} <class 'dict'> 2400159836224

>>> d1.pop(10)-------------'Apple'

>>> print(d1,type(d1),id(d1))----------{20: 'Mango', 30: 'Kiwi', 40: 'Banana'} <class 'dict'> 2400159836224

>>> d1.pop(40)-----------'Banana'

>>> print(d1,type(d1),id(d1))-----------{20: 'Mango', 30: 'Kiwi'} <class 'dict'> 2400159836224

>>> d1.pop(50)---------KeyError: 50

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

3) popitem()

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

=>This Function is used for removing last (key,value) from dict object.

=>Syntax:- dictobj.popitem()

=>When we call popitem() upon dictobject then we get KeyError.

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

Examples:

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

>>> d1={10:"Apple",20:"Mango",30:"Kiwi",40:"Banana"}

>>> print(d1,type(d1),id(d1))----{10: 'Apple', 20: 'Mango', 30: 'Kiwi', 40: 'Banana'} <class 'dict'> 2400155649792

>>> d1.popitem()------------(40, 'Banana')

>>> print(d1,type(d1),id(d1))-----{10: 'Apple', 20: 'Mango', 30: 'Kiwi'} <class 'dict'> 2400155649792

>>> d1.popitem()-----------(30, 'Kiwi')

>>> print(d1,type(d1),id(d1))---------{10: 'Apple', 20: 'Mango'} <class 'dict'> 2400155649792

>>> d1.popitem()--------(20, 'Mango')

>>> print(d1,type(d1),id(d1))--------{10: 'Apple'} <class 'dict'> 2400155649792

>>> d1.popitem()-----------(10, 'Apple')

>>> print(d1,type(d1),id(d1))----------{} <class 'dict'> 2400155649792

>>> d1.popitem()--------KeyError: 'popitem(): dictionary is empty'

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

4) copy()

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

=>This Function is used for copying the content of one dict object into another dict object.

=>Syntax: dictobj2=dictobj1.copy()

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

Examples:

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

>>> d1={10:"Apple",20:"Mango",30:"Kiwi"}

>>> print(d1,type(d1),id(d1))-----{10: 'Apple', 20: 'Mango', 30: 'Kiwi'} <class 'dict'> 2400159836224

>>> d2=d1.copy() # Shallow Copy

>>> print(d2,type(d2),id(d2))---{10: 'Apple', 20: 'Mango', 30: 'Kiwi'} <class 'dict'> 2400155649792

>>> d1[10]="Banana"

>>> d2[30]="Orange"

>>> print(d1,type(d1),id(d1))---{10: 'Banana', 20: 'Mango', 30: 'Kiwi'} <class 'dict'> 2400159836224

>>> print(d2,type(d2),id(d2))---{10: 'Apple', 20: 'Mango', 30: 'Orange'} <class 'dict'> 2400155649792

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

5) get()

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

=>This function is used for obtaining value of Value by passing value of Key.

=>If the value of Key does not exist then we get None

=>Syntax: varname=dictobj.get(Key)

Examples:

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

>>> d1={10:"Apple",20:"Mango",30:"Kiwi"}

>>> print(d1)----------------{10: 'Apple', 20: 'Mango', 30: 'Kiwi'}

>>> val=d1.get(10)

>>> print(val)----------------------Apple

>>> val=d1.get(20)

>>> print(val)--------------------Mango

>>> val=d1.get(40)

>>> print(val)--------------------None

(OR)

=>Instead of Using get(), we can also obtain value of Value by passing Val of Key by using the following Syntax:

varname=dictobj[Key]

=>If the key does not exist then we get KeyError.

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

Examples:

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

>>> d1={10:"Apple",20:"Mango",30:"Kiwi"}

>>> print(d1)---------------{10: 'Apple', 20: 'Mango', 30: 'Kiwi'}

>>> val=d1[10]

>>> print(val)----------------Apple

>>> val=d1[30]

>>> print(val)----------------Kiwi

>>> val=d1[40]------------KeyError: 40

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

6) keys()

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

=>This Function is used for obtaining values of Key.

=>Syntax: varname=dictobj.keys()

(OR)

dictobj.keys()

Examples:

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

>>> d1={10:"Apple",20:"Mango",30:"Kiwi"}

>>> print(d1)

{10: 'Apple', 20: 'Mango', 30: 'Kiwi'}

>>> ks=d1.keys()

>>> print(ks)-----------dict\_keys([10, 20, 30])

>>> for k in ks:

... print(k)

...

10

20

30

>>> for k in d1.keys():

... print(k)

...

10

20

30

NOTE:

-----------

>>> d1={10:"Apple",20:"Mango",30:"Kiwi"}

>>> print(d1)

{10: 'Apple', 20: 'Mango', 30: 'Kiwi'}

>>> for k in d1:

... print(k)

...

10

20

30

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

7) values()

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

=>This Function is used for obtaining values of Value.

=>Syntax: varname=dictobj.values()

(OR)

dictobj.values()

Examples:

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

>>> d1={10:"Apple",20:"Mango",30:"Kiwi"}

>>> print(d1)---------------{10: 'Apple', 20: 'Mango', 30: 'Kiwi'}

>>> vs=d1.values()

>>> print(vs)------------dict\_values(['Apple', 'Mango', 'Kiwi'])

>>> for v in vs:

... print(v)

...

Apple

Mango

Kiwi

>>> for vals in d1.values():

... print(vals)

...

Apple

Mango

Kiwi

NOTE:

-----------

>>> d1={10:"Apple",20:"Mango",30:"Kiwi"}

>>> print(d1)

{10: 'Apple', 20: 'Mango', 30: 'Kiwi'}

>>> for k in d1:

... print(k)

...

10

20

30

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

8) items()

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

=>This Function is used for obtaining (Key,value) in the form list of tuples.

=>Syntax: varname=d1.items()

Examples:

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

>>> d1={10:"Apple",20:"Mango",30:"Kiwi"}

>>> print(d1)-------------------{10: 'Apple', 20: 'Mango', 30: 'Kiwi'}

>>> kv=d1.items()

>>> print(kv)------------dict\_items([(10, 'Apple'), (20, 'Mango'), (30, 'Kiwi')])

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

>>> for kvs in kv:

... print(kvs)

...

(10, 'Apple')

(20, 'Mango')

(30, 'Kiwi')

>>> for h,b in kv:

... print(h,"--->",b)

...

10 ---> Apple

20 ---> Mango

30 ---> Kiwi

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

>>> for k,v in d1.items():

... print(k,"<--->",v)

...

10 <---> Apple

20 <---> Mango

30 <---> Kiwi

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

9) update()

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

=>Syntax: dictobj1.update(dictobj2)

=>This function is used for updating the old values of dictobj1 with New Values of dictobj2(Either Insertion or Modification)

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

Examples:

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

>>> d1={10:3.4,20:5.6,30:6.7}

>>> print(d1)-------------------------{10: 3.4, 20: 5.6, 30: 6.7}

>>> d2={40:5.6,50:6.7,55:1.2}

>>> print(d2)-----------------------{40: 5.6, 50: 6.7, 55: 1.2}

>>> d1.update(d2)

>>> print(d1)----------------------{10: 3.4, 20: 5.6, 30: 6.7, 40: 5.6, 50: 6.7, 55: 1.2}

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

>>> d1={10:3.4,20:5.6,30:6.7}

>>> print(d1)-----------------------{10: 3.4, 20: 5.6, 30: 6.7}

>>> d2={10:15.6,20:16.7,30:1.2}

>>> print(d2)----------------------{10: 15.6, 20: 16.7, 30: 1.2}

>>> d1.update(d2)

>>> print(d1)------------------{10: 15.6, 20: 16.7, 30: 1.2}

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

>>> d1={10:3.4,20:5.6,30:6.7}

>>> print(d1)---------------------------{10: 3.4, 20: 5.6, 30: 6.7}

>>> d2={10:12.3,15:1.2,25:4.5}

>>> print(d2)---------------------------{10: 12.3, 15: 1.2, 25: 4.5}

>>> d1.update(d2)

>>> print(d1)--------------------{10: 12.3, 20: 5.6, 30: 6.7, 15: 1.2, 25: 4.5}

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

Misc Examples:

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

>>> l1=[(10,"Rossum"),(20,"Ritche"),(30,"Travis"),(40,"Oliphant")]

>>> print(l1,type(l1))---[(10, 'Rossum'), (20, 'Ritche'), (30, 'Travis'), (40, 'Oliphant')] <class 'list'>

>>> d1=dict(l1)

>>> print(d1,type(d1))---{10: 'Rossum', 20: 'Ritche', 30: 'Travis', 40: 'Oliphant'} <class 'dict'>

>>> for ano,an in d1.items():

... print(ano,"-->",an)

...

10 --> Rossum

20 --> Ritche

30 --> Travis

40 --> Oliphant

Examples:

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

>>> l2=list(d1)

>>> print(l2)------------------[10, 20, 30, 40]

>>> l2=list(d1.items())

>>> print(l2)-----------[(10, 'Rossum'), (20, 'Ritche'), (30, 'Travis'), (40, 'Oliphant')]

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

Misc Example--IMP

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

>>> d1={1:["Python","DSc","Django"],

... 2:("Java","AdvJava","HiberNate","Spring"),

... 3:{"HTML","CSS","JS","VS"} }

>>>

>>> print(d1,type(d1))

{1: ['Python', 'DSc', 'Django'], 2: ('Java', 'AdvJava', 'Hibernate', 'Spring'), 3: {'JS', 'VS', 'CSS', 'HTML'}} <class 'dict'>

>>> for k,v in d1.items():

... print(k,"<--->",v)

...

1 <---> ['Python', 'DSc', 'Django']

2 <---> ('Java', 'AdvJava', 'Hibernate', 'Spring')

3 <---> {'JS', 'VS', 'CSS', 'HTML'}

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

MISC Example----enumerate()

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

>>> s="Apple is Red"

>>> x=enumerate(s)

>>> print(x, type(x))

<enumerate object at 0x000002A9B277BA00> <class 'enumerate'>

>>> for ci,cn in x:

... print(ci,"--->",cn)

...

0 ---> A

1 ---> p

2 ---> p

3 ---> l

4 ---> e

5 --->

6 ---> i

7 ---> s

8 --->

9 ---> R

10 ---> e

11 ---> d

>>> for cn,ci in enumerate("Hyderabad"):

... print(ci,"--->",cn)

...

H ---> 0

y ---> 1

d ---> 2

e ---> 3

r ---> 4

a ---> 5

b ---> 6

a ---> 7

d ---> 8

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

MiSc Example----zip()

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

>>> l1=[10,20,30,40]

>>> l2=["Rossum","Ritche","Travis","Oliphant"]

>>> z=zip(l1,l2)

>>> print(z, type(z))

<zip object at 0x000002A9B277BC00> <class 'zip'>

>>> for rno,rn in z:

... print(rno,"--->",rn)

...

10 ---> Rossum

20 ---> Ritche

30 ---> Travis

40 ---> Oliphant

>>> l1=[10,20,30,40]

>>> l2=["Rossum","Ritche","Travis","Oliphant","KVR"]

>>> z=zip(l1,l2)

>>> for rno,rn in z:

... print(rno,"--->",rn)

...

10 ---> Rossum

20 ---> Ritche

30 ---> Travis

40 ---> Oliphant

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