**BASIC PYTHON PROGRAMS**

**Printing statements in Python:** Python print () Function The print() function prints the specified message to the screen, or other standard output device.

**Program:**

**# Printing a string**

print("Hello, Everyone")

print('1,2,3,4')

a = 5

print('The value of a is', a)

print(1, 2, 3, 4)

print(1, 2, 3, 4, sep='\*')

print(1, 2, 3, 4, sep='#', end='&')

**Output:**

Hello, Everyone

1,2,3,4

The value of a is 5

1 2 3 4

1\*2\*3\*4

1#2#3#4&

**Python Indentation**

Indentation refers to the spaces at the beginning of a code line. Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important. Python uses indentation to indicate a block of code.

**Example :**

if 5 > 2:  
  print("Five is greater than two!")

**Output:** Five is greater than two!

**Variables:** Variable is a name that is used to refer to memory location. Python variable is also known as an identifier and used to hold value.

In Python, we don't need to specify the type of variable because Python is a infer language and smart enough to get variable type.

Variable names can be a group of both the letters and digits, but they have to begin with a letter or an underscore.

**Rules:**

* The first character of the variable must be an alphabet or underscore ( \_ ).
* All the characters except the first character may be an alphabet of lower-case(a-z), upper-case (A-Z), underscore, or digit (0-9).
* Identifier name must not contain any white-space, or special character (!, @, #, %, ^, &, \*).
* Identifier name must not be similar to any keyword defined in the language.
* Identifier names are case sensitive; for example, my name, and MyName is not the same.
* Examples of valid identifiers: a123, \_n, n\_9, etc.
* Examples of invalid identifiers: 1a, n%4, n 9, etc.

**Program-1:**

var1 = 10 # An integer assignment

var2 = 3.146 # A floating point

var3 = "Hello" # A string

print(var1,' ',var2,' ',var3)

**Output:** 10 3.146 Hello

**Program-2:**

# Assigning same value to multiple variables

var1 = var2 = var3 = 1

print(var1,' ',var2,' ',var3)

# Assigning Different values to variable in a single expression

var1, var2, var3 = 1, 2.5, "AI TOOLS"

print(var1,' ',var2,' ',var3)

# Note: commas can be used for multi-assignments

**Output:**

1 1 1

1 2.5 AI TOOLS

**Python User Input from Keyboard – input () function**

* Python user input from the keyboard can be read using the input () built-in function.
* The input from the user is read as a string and can be assigned to a variable.
* After entering the value from the keyboard, we have to press the “Enter” button. Then the input () function reads the value entered by the user.

**The syntax of input () function is:**

**input(prompt)**

**Program:**

num = input ("Enter number :")

print(num)

num1 = float(input("Enter the floating point Number:"))

print(num1)

name1 = input("Enter name : ")

print(name1)

**Output:**

Enter number: 12

12

Enter the floating point Number: 3.5

3.5

Enter name: AI TOOLS LAB

AI TOOLS LAB

**Datatypes:**

Data types are the classification or categorization of data items. Data types represent a kind of value which determines what operations can be performed on that data. Variables can hold values, and every value has a data-type. Python is a dynamically typed language; hence we do not need to define the type of the variable while declaring it. The interpreter implicitly binds the value with its type.

**For eg: a=5**

* The variable a holds integer value five and we did not define its type. Python interpreter will automatically interpret variables a as an integer type.
* Python enables us to check the type of the variable used in the program. Python provides us the type() function, which returns the type of the variable passed.
* Python provides various standard data types that define the storage method on each of them.
* **The data types defined in Python are given below.**

1. [Numbers](https://www.javatpoint.com/python-data-types#numbers)
2. [Sequence Type](https://www.javatpoint.com/python-data-types#SequenceType)
3. [Boolean](https://www.javatpoint.com/python-data-types#Boolean)
4. [Set](https://www.javatpoint.com/python-data-types#Set)
5. [Dictionary](https://www.javatpoint.com/python-data-types#dictionary)

**1. Numeric:** Number stores numeric values. The integer, float, and complex values belong to a Python Numbers data-type. Python provides the **type()** function to know the data-type of the variable. Similarly, the **isinstance()** function is used to check an object belongs to a particular class.

**Program-**1:

a = 5

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

b = 40.5

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

c = 1+3j

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

print(" c is a complex number", isinstance(1+3j,complex)) ‘

**Output:**

The type of a <class 'int'>

The type of b <class 'float'>

The type of c <class 'complex'>

c is a complex number True

**2. Sequence Type:**

**(a) String:** The string can be defined as the sequence of characters represented in the quotation marks. In Python, we can use single, double, or triple quotes to define a string.

**Program:**

**# String operations**

str = 'Hello World!' # A string

print(str) # Prints complete string

print(str[0]) # Prints first character of the string

print(str[2:5]) # Prints characters starting from 3rd to 5th element

print(str[2:]) # Prints string starting from 3rd character

print(str \* 2) # Prints string two times

print(str + "TEST") # Prints concatenated string

**Output:**

Hello World!

H

llo

llo World!

He

Hello World!Hello World!

Hello World!TEST

**(b) List:** List is a collection which is **ordered and changeable**. Allows **duplicate members.** Python Lists are similar to arrays in C. However, the list can contain data of different types. The items stored in the list are separated with a comma (,) and enclosed within square brackets [].

**Program:**

**# Create a List**

thislist = ["apple", "banana", "cherry"]

print(thislist)

**# Access Elements**

thislist = ["apple", "banana", "cherry"]

print(thislist[1])

**#range of Indexes**

thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]

print(thislist[2:5]) # here prints only 2,3,4 positions. 5th position not included

**# Change the Item Value**

thislist = ["apple", "banana", "cherry"]

print(thislist)

thislist[1] = "blackcurrant"

print(thislist)

**# add the items in the list**

thislist.append("orange")

print(thislist)

**# Remove an items in the list**

thislist = ["apple", "banana", "cherry"]

thislist.remove("banana")

print(thislist)

**# Join two Lists**

list1 = ["a", "b" , "c"]

list2 = [1, 2, 3]

list3 = list1 + list2

print(list3)

**Output:**

['apple', 'banana', 'cherry']

banana

['cherry', 'orange', 'kiwi']

['apple', 'banana', 'cherry']

['apple', 'blackcurrant', 'cherry']

['apple', 'blackcurrant', 'cherry', 'orange']

['apple', 'cherry']

['a', 'b', 'c', 1, 2, 3]

**(c) Tuple:** is a collection which is ordered and **unchangeable. Allows duplicate members.**

**Program:**

**# Create a tuple**

thistuple = ("Maths", "Physics", "Chemistry")

print(thistuple)

**# Print 1st item in the tuple**

print(thistuple[1])

**# add the items in the tuple**

thistuple.append("computer") # error , because tuples are immutable.

print(thistuple)

**Output:**

('Maths', 'Physics', 'Chemistry')

Physics

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

AttributeError Traceback (most recent call last)

**(d) Dictionary**: Python Dictionary is used to **store the data in a key-value pair format**. The dictionary is the data type in Python, which can simulate the **real-life data arrangement** where some specific value exists for **some particular key**. It is the **mutable data-structure.** The dictionary is defined into element **Keys and values.**

* Keys must be a single element
* Value can be any type such as list, tuple, integer, etc.

**Program:**

**# Creating a Dictionary**

Employee = {"Name": "John", "Age": 29, "salary":25000,"Company":"GOOGLE"}

print(type(Employee))

print("printing Employee data .... ")

print(Employee)

**# Creating an empty Dictionary using Dict function. it is one of the built in function in Python**

Dict = {}

print("Empty Dictionary: ")

print(Dict)

**# Creating a Dictionary with dict() method**

Dict = dict({1: 'Python', 2: 'Programming', 3:'Language'})

print("\nCreate Dictionary by using dict(): ")

print(Dict)

**# uisng for loop**

Employee = {"Name": "John", "Age": 29, "salary":25000,"Company":"GOOGLE"}

for x in Employee:

print(x)

**#for loop to print all the values of the dictionary**

Employee = {"Name": "John", "Age": 29, "salary":25000,"Company":"GOOGLE"}

for x in Employee:

print(Employee[x])

**Output:**

<class 'dict'>

printing Employee data ....

{'Name': 'John', 'Age': 29, 'salary': 25000, 'Company': 'GOOGLE'}

Empty Dictionary:

{}

Create Dictionary by using dict():

{1: 'Python', 2: 'Programming', 3: 'Language'}

Name

Age

salary

Company

John

29

25000

GOOGLE

**Built in Functions:**

**Program:**

print("Sum of array: ",sum([1,2,3,4]))

print("Length of array: ",len([1,2,3,4]))

print("Absolute value: ",abs(-1234))

print("Round value: ",round(1.2234))

import math as mt # importing a package

print("Log value: ",mt.log(10))

**Output:**

Sum of array: 10

Length of array: 4

Absolute value: 1234

Round value: 1

Log value: 2.302585092994046

**Functions:**

**Program:**

def area(length,width):

return length\*width

are = area(10,20)

print("Area of rectangle:",are)

**Output:** Area of rectangle: 200

**NUMPY:** NumPy stands for **Numerical Python.** NumPy is a **python library used for working with arrays**. It also has functions for working in domain of **linear algebra, fourier transform, and matrices.** It is an open source project and you can use it freely.

**Program:**

import numpy as np # Importing libraries

a = np.array([0, 1, 2])

b = np.array([5, 5, 5])

print("Matrix A\n", a)

print("Matrix B\n", b)

print("Regular matrix addition A+B\n", a + b)

print("Addition using Broadcasting A+5\n", a + 5)

**Output:**

Matrix A

[0 1 2]

Matrix B

[5 5 5]

Regular matrix addition A+B

[5 6 7]

Addition using Broadcasting A+5

[5 6 7]

### Broadcasting Rules

When operating on two arrays, NumPy compares their shapes element-wise. It starts with the trailing dimensions, and works its way forward. Two dimensions are compatible when

1. they are equal, or
2. one of them is 1

**Program-1:**

**# Lets go for a 2D matrix**

c = np.array([[0, 1, 2],[3, 4, 5],[6, 7, 8]])

d = np.array([[1, 2, 3],[1, 2, 3],[1, 2, 3]])

e = np.array([1, 2, 3])

print("Matrix C\n", c)

print("Matrix D\n", d)

print("Matrix E\n", e)

print("Regular matrix addition C+D\n", c + d)

print("Addition using Broadcasting C+E\n", c + e)

**Output:**

Matrix C

[[0 1 2]

[3 4 5]

[6 7 8]]

Matrix D

[[1 2 3]

[1 2 3]

[1 2 3]]

Matrix E

[1 2 3]

Regular matrix addition C+D

[[ 1 3 5]

[ 4 6 8]

[ 7 9 11]]

Addition using Broadcasting C+E

[[ 1 3 5]

[ 4 6 8]

[ 7 9 11]]

**Program-2:**

M = np.ones((3, 3))

print("Matrix M:\n",M)

**Output:**

Matrix M:

[[1. 1. 1.]

[1. 1. 1.]

[1. 1. 1.]]

## Essential Python Packages: Numpy, Pandas, Matplotlib

**Program:**

**# Load library**

import numpy as np

**# Create matrix**

matrix = np.array([[1, 2, 3],

[4, 5, 6],

[7, 8, 9]])

print("Matrix\n",matrix)

**# Select second row**

print("Second row of Matrix\n",matrix[1,:])

print("Third coloumn of Matrix\n",matrix[:,2])

**Output:**

Matrix

[[1 2 3]

[4 5 6]

[7 8 9]]

Second row of Matrix

[4 5 6]

Third coloumn of Matrix

[3 6 9]

**Matrix Properties:**

**Program:**

**# Create matrix**

matrix = np **Program:**

.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

print("Matrix Shape:",matrix.shape)

print("Number of elements:",matrix.size)

print("Number of dimentions:",matrix.ndim)

print("Average of matrix:",np.mean(matrix))

print("Maximum number:",np.max(matrix))

print("Coloumn with minimum numbers:",np.min(matrix, axis=1))

print("Diagnol of matrix:",matrix.diagonal())

**Output:**

Matrix Shape: (3, 3)

Number of elements: 9

Number of dimentions: 2

Average of matrix: 5.0

Maximum number: 9

Coloumn with minimum numbers: [1 4 7]

Diagnol of matrix: [1 5 9]

**Matrix Operations**

**Program:**

### print("Flattened Matrix\n",matrix.flatten())

### print("Reshaping Matrix\n",matrix.reshape(9,1))

### print("Transposed Matrix\n",matrix.T)

**Output:**

Flattened Matrix

[1 2 3 4 5 6 7 8 9]

Reshaping Matrix

[[1] [2] [3]

[4]

[5]

[6]

[7]

[8]

[9]]

Transposed Matrix

[[1 4 7]

[2 5 8]

[3 6 9]]

**Multiply Two Matrices:**

**Program:**

**# Create a matrix**

matrix\_a = np.array([[1, 1, 1],

[1, 1, 1],

[1, 1, 2]])

**# Create b matrix**

matrix\_b = np.array([[1, 3, 1],

[1, 3, 1],

[1, 3, 8]])

print("Matrix Addition\n",np.add(matrix\_a, matrix\_b))

print("Scalar Multiplication\n",np.multiply(matrix\_a, matrix\_b))

**Output:**

Matrix Addition

[[ 2 4 2]

[ 2 4 2]

[ 2 4 10]]

Scalar Multiplication

[[ 1 3 1]

[ 1 3 1]

[ 1 3 16]]