**\*Python for Beginners \***

**\*Core Python \* \*Advanced Python\***

• Class & Objects

• OOPs

• RegX Expression

• CGI Programming

• DataBase Access

• Inheritance

• Encapsulation

• Polymorphism

• Data Abstaraction

• Operator Overloading

• Polymorphism

• ITERATORS & Generators

• CLOSURES & DECORATORS

* Python Overview (Introduction)
* Environment Setup
* Basic syntax
* Comments
* Variable/Casting
* Basic Operators
* Data Types (Data structures)
* Indentation
* Condition/Decision Making
* Loops
* Function
* Lambda
* List
* Tuple
* Dictionary
* Set
* Range
* Module

Python IDE & Code Editors:- (Integrated Development Editors)

• Jupyter notebook • Visual studio code

• Eclipse • Azure Data studio

• IntelliJ • Spyder

• PyCharm • PyDev

\*Python Libraries/Frameworks \*

* Numpy - Scientific computing library
* Pandas - Data science library
* Matplotlib - Data Visualization library
* SciPy - Data visualization & manipulation librarySCrapy - Web Scraping framework
* TensorFlow - Deep Learning & Machine Learning Library
* Flask - Web Development
* Django - Web Development
* Tkinter - GUI development Library
* Pygame - 2D game development library
* Theano - Machine & deep Learning Library
* Pytorch - Data Science Library
* Pillow - Image Manipulation Library

**\*Career options in Python\* :-**

•Data engineer •Data scientist •Data analyst •DevOps engineer

•Python developer •Game developer •Web designer •Full-stack developer

•Machine learning engineer •Software engineer & Many more other roles

* **Python Overview (Introduction) :-**

Python was created by Guido van Rossum, and released in 1991 and It is one of the most popular programming languages.it is used in various areas of applications such as Machine Learning, Artificial Intelligence, web development, IoT, and more.P ython works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc). The most recent major version of Python is Python 3.

* **Python Installation (Environment Setup) :-**

It can be downloaded from its official website [Download Python | Python.org](https://www.python.org/downloads/)

A screenshot of a computer

Description automatically generated

**To check Python Version :-** python --version

Program Execution Mode

* Interactive Mode :- print("Hello, World!")
* Scripting Mode :- C:\Users\S9WNDR\Desktop\Python train Env>python sum.py

Basic Syntax :-

1 + 1 #2

2 - 1 #1

2 \* 2 #4

4 / 2 #2

4 % 3 #1

4 \*\* 2 #16

4 // 2 #2

1 + 2

1 + 2 ; 3 + 5

print("My first python code")

Comments in Python (#):- Everything after a hash mark is ignored, and considered a comment:

print("Hello, Python3")#This is a comment

#print("Hello Aman")  
 print("Data engineer")#Multi Line comments not supported in python

#This is a comment  
#written in  
#more than just one line  
print("Hello, World!")

Since Python will ignore string literals that are not assigned to a variable, you can add a multiline string (triple quotes) in your code, and place your comment inside it:

"""  
This is a comment  
written in  
more than just one line  
"""  
print("Hello, World!")

**Python Variable / Casting :-**

Python has no command for declaring a variable but it is created at the moment when it assign some value to it. It reserved memory locations used to store values with in a Python Program. This means that when we create a variable its reserve some space in the memory.by assigning different data types to Python variables, It can store integers, decimals or characters in these variables.

The equal sign (=) is used to assign values to variables.

Example :-

Kilometer = 250 # Creates an integer variable

miles = 1034.0 # Creates a floating point variable

name = "Python3" # Creates a string variable

String variables can be declared either by using single or double quotes:

x = "John"  
# is the same as  
x = 'John'

**Printing the declared variable name :-**

print (Kilometer)

print (miles)

print (name)

**Deleting Python Variables**

del var1[,var2[,var3[....,varN]]]]

del var

del var\_a, var\_b

counter = 100

print (counter)

del counter

print (counter)

**Getting Type of a Variable :- we can get the data type of a variable with the type() function.**

x = "Zara"

y = 10

z = 10.10

print(type(x))

print(type(y))

print(type(z))

**Casting Python Variables**

You can specify the data type of a variable with the help of casting as follows:

x = str(10) # x will be '10'

y = int(10) # y will be 10

z = float(10) # z will be 10.0

print( "x =", x )

print( "y =", y )

print( "z =", z )

**Case-Sensitivity of Python Variables**

Python variables are case sensitive which means Age and age are two different variables:

age = 20

Age = 30

print( "age =", age )

print( "Age =", Age )

**Memory Addresses**

Data items belonging to different data types are stored in computer's memory. Computer's memory locations are having a number or address, internally represented in binary form. Python's built-in id() function returns the address where the object is stored.

>>> "May"

May

>>> id("May")

2167264641264

>>> 18

18

>>> id(18)

140714055169352

**Variables - Multiple Assignment**

>>> a=10

>>> b=10

>>> c=10

>>> a=b=c=10

>>> print (a,b,c)

In the following case, we have three variables with different values.

>>> a=10

>>> b=20

>>> c=30

>>> a,b,c = 10,20,30

>>> print (a,b,c)

a = b = c = 100

print (a)

print (b)

print (c)

a,b,c = 1,2,"Zara Ali"

print (a)

print (b)

print (c)

Variables - Naming Convention

A variable name must start with a letter or the underscore character

A variable name cannot start with a number or any special character like $, (, \* % etc.

A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and \_ )

Python variable names are case-sensitive which means Name and NAME are two different variables in Python.

Python reserved keywords cannot be used naming the variable.

counter = 100

count = 100

name1 = "Zara"

name2 = "Nuha"

Age = 20

zara\_salary = 100000

print (counter)

print (\_count)

print (name1)

print (name2)

print (Age)

print (zara\_salary)

Invalid Variable Name:-

1counter = 100

$\_count = 100

zara-salary = 100000

print (1counter)

print ($count)

print (zara-salary)

width = 10

height = 20

area = width\*height

perimeter = 2\*(width+height)

print ("Area = ", area)

print ("Perimeter = ", perimeter)

**Python Local Variable & Global Variable :-**

Python Local Variables are defined inside a function and Global variable define outside the function.

Example :-

Global

x = 5

y = 10

def sum():

sum = x + y

return sum

print(sum())

Local

def sum(x,y):

sum = x + y

return sum

print(sum(5, 10))

**Basic Operators :-**

Python operators are special symbols (sometimes called keywords) that are used to perform certain most commonly required operations on one or more operands (value, variables, or expressions).

Arithmetic Operators ()

Comparison (Relational) Operators

Assignment Operators

Logical Operators

Bitwise Operators

Membership Operators

Identity Operators

**Arithmetic Operators ()**

+ Addition x + y

- Subtraction x - y

\* Multiplication x \* y

/ Division x / y

% Modulus x % y

\*\* Exponentiation x \*\* y

// Floor division x // y

**Comparison (Relational) Operators**

== Equal (a == b) is not true.

!= Not equal (a != b) is true.

> Greater than (a > b) is not true.

< Less than (a < b) is true.

>= Greater than or equal to (a >= b) is not true.

<= Less than or equal to (a <= b) is true.

**Assignment Operators**

= a = 10 a = 10

+= a += 30 a = a + 30

-= a -= 15 a = a - 15

\*= a \*= 10 a = a \* 10

/= a /= 5 a = a / 5

%= a %= 5 a = a % 5

\*\*= a \*\*= 4 a = a \*\* 4

//= a //= 5 a = a // 5

&= a &= 5 a = a & 5

|= a |= 5 a = a | 5

^= a ^= 5 a = a ^ 5

>>= a >>= 5 a = a >> 5

<<= a <<= 5 a = a << 5

**Logical Operators**

Operator Name Example

and AND a and b

or OR a or b

not NOT not(a)

example :-

var = 5

print(var > 3 and var < 10)

print(var > 3 or var < 4)

print(not (var > 3 and var < 10))

**Bitwise Operators**

Operator Name Example

& AND a & b

| OR a | b

^ XOR a ^ b

~ NOT ~a

<< Zero fill left shift a << 3

>> Signed right shift a >> 3

Membership Operators

Python's membership operators test for membership in a sequence, such as strings, lists, or tuples. There are two membership operators as explained below −

A screenshot of a computer

Description automatically generated

a = 10

b = 20

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

print ("a:", a, "b:", b, "list:", list)

if ( a in list ):

print ("a is present in the given list")

else:

print ("a is not present in the given list")

**Identity Operators**

Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:

A screenshot of a computer

Description automatically generated

Example :-

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

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

c = a

print(a is c)

print(a is b)

print(a is not c)

print(a is not b)

* **Data Types (Data structures)**

A data type represents a kind of value and determines what operations can be done on it. Numeric, non-numeric and Boolean (true/false) data are the most obvious data types.

A screenshot of a data

Description automatically generated

A screenshot of a computer

Description automatically generated

Numeric :-

var1 = 1 # int data type

var2 = True # bool data type

var3 = 10.023 # float data type

var4 = 10+3j # complex data type

a=100

print("The type of variable having value", a, " is ", type(a))

# float variable.

c=20.345

print("The type of variable having value", c, " is ", type(c))

# complex variable.

d=10+3j

print("The type of variable having value", d, " is ", type(d))

String :-

Python string is a sequence of one or more Unicode characters, enclosed in single, double or triple quotation marks (also called inverted commas).

>>> 'swissre'

'swissre'

>>> "swissre"

'swissre'

>>> '''swissre'''

'swissre'

>>> >>> type("Welcome To Swiss reinsurance")

<class 'str'>

A string is a non-numeric data type. Obviously, we cannot perform arithmetic operations on it. However, operations such as slicing and concatenation can be done. Python's str class defines a number of useful methods for string processing. Subsets of strings can be taken using the slice operator ([ ] and [:] ) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.

The plus (+) sign is the string concatenation operator and the asterisk (\*) is the repetition operator in Python

str = 'Welcome to swissre!'

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

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

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

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

**Sequence :-**

Sequence is a collection data type. It is an ordered collection of items. Items in the sequence have a positional index starting with 0. It is conceptually similar to an array in C or C++. There are following three sequence data types defined in Python.

**List :- Denoted by []**

list = [ 'abcd', 786 , 2.23, 'swissre', 70.2 ]

mylist = [123, 'swissre']

print (list) # Prints complete list

print (list[0]) # Prints first element of the list

print (list[1:3]) # Prints elements starting from 2nd till 3rd

print (list[2:]) # Prints elements starting from 3rd element

print (tinylist \* 2) # Prints list two times

print (list + mylist) # Prints concatenated lists

**Tuple :- Denoted by ()**

tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )

Mytuple = (123, 'john')

print (tuple) # Prints the complete tuple

print (tuple[0]) # Prints first element of the tuple

print (tuple[1:3]) # Prints elements of the tuple starting from 2nd till 3rd

print (tuple[2:]) # Prints elements of the tuple starting from 3rd element

print (tinytuple \* 2) # Prints the contents of the tuple twice

print (tuple + Mytuple) # Prints concatenated tuples

**The main differences between lists and tuples are: Lists are enclosed in brackets ( [ ] ) and their elements and size can be changed i.e. lists are mutable, while tuples are enclosed in parentheses ( ( ) ) and cannot be updated (immutable). Tuples can be thought of as read-only lists.**

**The following code is invalid with tuple, because we attempted to update a tuple, which is not allowed. Similar case is possible with lists –**

tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )

list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]

tuple[2] = 1000 # Invalid syntax with tuple

list[2] = 1000 # Valid syntax with list

**Range ()**

A Python range is an immutable sequence of numbers which is typically used to iterate through a specific number of items.

It is represented by the Range class. The constructor of this class accepts a sequence of numbers starting from 0 and increments to 1 until it reaches a specified number. Following is the syntax of the function −

range(start, stop, step)

Here is the description of the parameters used −

start: Integer number to specify starting position, (Its optional, Default: 0)

stop: Integer number to specify ending position (It's mandatory)

step: Integer number to specify increment, (Its optional, Default: 1)

Example :-

for i in range(5):

print(i)

for i in range(2, 5):

print(i)

for i in range(1, 5, 2):

print(i)

**Python dictionaries** :- It is a kind of hash table type. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.

It consist of key:value pairs. The pairs are separated by comma and put inside curly brackets {}. To establish mapping between key and value, the semicolon':' symbol is put between the two.

Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]).

Example :-

dict = {}

dict['one'] = "This is one"

dict[2] = "This is two"

tinydict = {'name': 'john','code':6734, 'dept': 'sales'}

print (dict['one']) # Prints value for 'one' key

print (dict[2]) # Prints value for 2 key

print (tinydict) # Prints complete dictionary

print (tinydict.keys()) # Prints all the keys

print (tinydict.values()) # Prints all the values

**Set:-**

A set in Python is a collection, but is not an indexed or ordered collection as string, list or tuple. An object cannot appear more than once in a set, whereas in List and Tuple, same object can appear more than once.

Comma separated items in a set are put inside curly brackets or braces {}. Items in the set collection can be of different data types.

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

Set is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Tuple, and Dictionary, all with different qualities and usage.

A set is a collection which is unordered, unchangeable\*, and unindexed.

myset = {"apple", "banana", "cherry"}

myset = {"apple", "banana", "cherry"}  
print(myset)

set1 = {123, 452, 5, 6}

set2 = {'Java', 'Python', 'JavaScript'}

print(set1)

print(set2)

**Python Boolean** :- It is a type one of built-in data types which represents one of the two values either True or False. Python bool() function allows you to evaluate the value of any expression and returns either True or False based on the expression.

# Returns false as a is not equal to b

a = 2

b = 4

print(bool(a==b))

# Following also prints the same

print(a==b)

# Returns False as a is None

a = None

print(bool(a))

# Returns false as a is an empty sequence

a = ()

print(bool(a))

# Returns false as a is 0

a = 0.0

print(bool(a))

# Returns false as a is 10

a = 10

print(bool(a))

**Type conversion :-**

print("Conversion to integer data type")

a = int(1) # a will be 1

b = int(2.2) # b will be 2

c = int("3.3") # c will be 3

1. Python int() function

Converts x to an integer. base specifies the base if x is a string.

1. Python long() function

Converts x to a long integer. base specifies the base if x is a string. This function has been deprecated.

1. Python float() function

Converts x to a floating-point number.

1. Python str() function

Converts object x to a string representation.

**Decision Making(IF….Else):-**

marks = 80

result = ""

if marks < 30:

result = "Failed"

elif marks > 75:

result = "Passed with distinction"

else:

result = "Passed"

print(result)

Python supports Match-Case statement, which can also be used as a part of decision making. If a pattern matches the expression, the code under that case will execute.

Example

def checkVowel(n):

match n:

case 'a': return "Vowel alphabet"

case 'e': return "Vowel alphabet"

case 'i': return "Vowel alphabet"

case 'o': return "Vowel alphabet"

case 'u': return "Vowel alphabet"

case \_: return "Simple alphabet"

print (checkVowel('a'))

print (checkVowel('m'))

print (checkVowel('o'))

**For loop:-**

words = ["one", "two", "three"]

for x in words:

print(x)

**while Loop:-**

i = 1

while i < 6:

print(i)

i += 1

**Break Statement:-**

x = 0

while x < 10:

print("x:", x)

if x == 5:

print("Breaking...")

break

x += 1

print("End")

**Function:-** **A Python function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.**

A diagram of a function

Description automatically generated

def greetings():

"This is docstring of greetings function"

print ("Hello World")

return

greetings()

# Function definition is here

def printme( str ):

"This prints a passed string into this function"

print (str)

return;

# Now you can call printme function

printme("I'm first call to user defined function!")

printme("Again second call to the same function")

**Argument function**

def greetings(name):

"This is docstring of greetings function"

print ("Hello {}".format(name))

return

greetings("Samay")

greetings("Pratima")

greetings("Steven")

total = 0; # This is global variable.

# Function definition is here

def sum( arg1, arg2 ):

# Add both the parameters and return them."

total = arg1 + arg2; # Here total is local variable.

print ("Inside the function local total : ", total)

return total;

# Now you can call sum function

sum( 10, 20 );

print ("Outside the function global total : ", total)

**Lambda Function :-** A lambda function is a small anonymous function.

A lambda function can take any number of arguments, but can only have one expression.

Syntax:- **lambda arguments : expression**

Example :-

x = lambda a, b : a \* b  
print(x(5, 6))

User Input :-

In a Python command line application you can display information to the user using the print() function:

name = "Roger"

print(name)

We can also accept input from the user, using input() :

print('What is your age?')

age = input()

print('Your age is ' + age)

A screenshot of a computer

Description automatically generated