Introduction of Python

Python

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. It was created by Guido van Rossum during 1985-1990.

Characteristics of Python

- Python is processed at runtime by the interpreter. You do not need to compile your program before executing it.
- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- It supports automatic garbage collection.
- It can be easily integrated with C, C++, ActiveX, CORBA, and Java.

History of Python

- Python was conceived in the late 1980s by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to the ABC language, capable of exception handling and interfacing with the Amoeba operating system.
- Its implementation began in December 1989. Van Rossum was the lead developer for the project.
- It was started firstly as a hobby project because he was looking for an interesting project to keep him occupied during Christmas.
- The language was finally released in 1991.
- Its main objective is to provide code readability and advanced developer productivity.
- When it was released it had more than enough capability to provide classes with inheritance, several core data types exception handling and functions.

Features of Python

- Python is a multi-paradigm programming language which supports ***Object-oriented*** programming and ***structured*** programming.
- It uses ***dynamic typing*** and a combination of reference counting.
- It uses a cycle-detecting ***garbage collector*** for memory management.
- It also features ***dynamic name resolution*** (late binding), which binds method and variable names during program execution.
- It has ***filter***, ***map***, and ***reduce functions***, list comprehensions, ***dictionaries***, sets, and generator expressions.
- It was designed to be ***highly extensible***.

Setting Path in Python

- Before start working with Python, a specific path setting is to required.
- Your Python program and executable code can reside in any directory of your system.
- The Path is set using the Environment Variable of My Computer properties:
- To set the path, we need to follow the following steps:

  Right click on My Computer ->Properties

  ->Advanced System setting ->Environment Variable ->New

  In Variable name write path and in Variable value copy the path C://Python(i.e., path where Python is installed). Click Ok ->Ok.

To check the version of installed Python:

In Unix/Linux/Fedora: $python

In Windows: C:\Users\Ranjan>python

For printing some values:

```
print("RCCIIT")              # it will print RCCIIT
print( 'RCCIIT')             #it will also print RCCIIT
print("'RCCIIT"')            # it will also print RCCIIT
print(" " "RCCIIT" " ")      #it will also print RCCIIT
```

- Python uses the + character to add a variable to another variable.

```
x =  "RCCIIT"
y =  "KOLKATA"
z =  x + y
print(z)        # it will print RCCIITKOLKATA
```

- For numbers, the + character works as a mathematical operator.

```
x = 10
y = 20
print(x + y)          # it will print 30
```

- If you try to combine a string and a number, It will give you an error.

```
x=5
y= "RCCIIT"
print(x + y)       # TypeError
```

Python Indentation

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

In C:

```
int main()
{
printf("RCCIIT\n");
    printf("COLLEGE");
return 0;
}
```

o/p: RCCIIT
COLLEGE

In python:

```
print("RCCIIT")
  print("COLLEGE")
```

o/p: Indentation Error

In python:

```
print("RCCIIT")
print("COLLEGE")
```

o/p: RCCIIT
COLLEGE

Syntax of Python

Python Variables

- In Python, variables are created when we assign a value to it.
- Python has no command for declaring a variable.
- A variable is created at the moment we first assign a value to it.

Example:

```
x = 5                    # x is of type int
print(x)
x =  ''RCCIIT ''         # x is now of type str
print(x)
```

Naming Convention of Variable Names

- A variable name must start with a letter or the underscore character.
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _).
- Variable names are case-sensitive.

Syntax of Python

Assign Value to Multiple Variables

- Python allows us to assign values to multiple variables in one line.
- Example:

```
x, y, z = "RCCIIT", "COLLEGE", 700015
print(x)
print(y)
print(z)
```

o/p:
```
RCCIIT
COLLEGE
700015
```

- You can assign the *same* value to multiple variables in one line.

```
x = y = z = "RCCIIT"
```

Comment Line in Python

Single Line Comment: Comments starts with a # symbol. Comments can be placed at the end of a line to ignore the rest of the line.

```
#This is a comment
print("Hello, World!")     #This is a comment
```

Multi Line Comments: Python does not have a syntax for multi line comments. To add a multiline comment you could insert a # for each line.

```
#This is a comment
#written in
#more than just one line
```

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
"""
```

Data Types in Python

Python has five standard data types −

- Numbers
- String
- List
- Tuple
- Dictionary

Python Numbers: Number data types store numeric values. Python supports four different numerical types −

- int (signed integers)
- long (long integers) (use a lowercase l or uppercase L with long)
- float (floating point real values)
- complex (complex numbers)

  (A complex number consists of an ordered pair of real floating-point numbers denoted by x + yj)

Data Type Conversion in Python

- **int(x [,base])** **#** Converts x to an integer, base specifies if x is string.

```
print(int('12',8))          # output 10
```

- **complex(real [,imag])** **#** Creates a complex number.

```
print(complex(5,8))      # output (5+8j)
```

- **str(x)** **#** Converts object x to a string representation.

```
print('RCCIIT'+str(10))   # output RCCIIT10
```

- **eval(str)** **#** Evaluates a string and returns an object.

```
print(eval("5+3*2"))        # output 11
```

- **tuple(s)** **#** Converts to a tuple.

```
print(tuple([2,3,4]))       # output   (2, 3, 4)
```

- **list(s)** **#** Converts to a list.

```
print(list((2,3,4)))          # output   [2, 3, 4]
```

- **chr(x)** **#** Converts an integer to a character.

```
print(chr(65))    # output   A
```

- **ord(c)** # Return ASCII value of a character.

```
print(ord('A'))     # output  65
```

Types of Operator in Python

Python language supports the following types of operators.

- Arithmetic Operators
- Comparison (Relational) Operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

Types of Operator in Python

Arithmetic Operators

- + Addition
- - Subtraction
- * Multiplication
- / Division
- % Modulus
- ** Exponent

  Performs exponential (Example: 2**4=16)
- // Floor Division

  The division of operands where the result is the quotient in which the digits after the decimal point are removed.

  (Example: 9//2 = 4 and 9.0//2.0 = 4.0, -11//3 = -4, -11.0//3 = -4.0)

Types of Operator in Python

Comparison (Relational) Operators

- == equal
- != not equal
- <> not equal
- > greater than
- < less than
- >= greater than or equal
- <= less than or equal

Python Bitwise Operators

- & Binary AND
- | Binary OR
- ^ Binary XOR
- ~ Binary Ones Complement
- << Binary Left Shift
- >> Binary Right Shift

Types of Operator in Python

Assignment Operators

- = Assignment — c = a
- += Addition and assign — c += a is equivalent to c = c + a
- -= Subtraction and assign — c -= a is equivalent to c = c – a
- *= Multiply and assign — c *= a is equivalent to c = c * a
- /= Divide and assign — c /= a is equivalent to c = c / a
- %= Modulus and assign — c %= a is equivalent to c = c % a
- **= Exponent and assign — c **= a is equivalent to c = c ** a
- //= Floor Division and assign — c //= a is equivalent to c = c // a

Logical Operators

- and Logical AND — # If both values are true then condition is true.
- or Logical OR — # If any one is non-zero then condition is true.
- not Logical NOT — # Reverse the logical state of its operand.

Types of Operator in Python

Membership Operators: Python's membership operators test for membership in a sequence, such as strings, lists, or tuples. There are two membership operators.

- in — **x in y**, result is true if x is a member of sequence y.
- not in — **x not in y**, result is true if x is not a member of sequence y.

Identity Operators: Identity operators compare the memory locations of two objects. There are two Identity operators.

- is — **x is y**, result in 1 if id(x) equals id(y).
- is not — **x is not y**, results in 1 if id(x) is not equal to id(y).

Python program to illustrate the use of 'is' identity operator

```
x = 5
if (type(x) is int):
    print("true")
else:
    print("false")
```

#Display float number with 2 decimal places using print()

```
x=458.541315
y=10
print('%.2f' %x )
print('%.2f' %x, '%d' %y)
print('Result=', '%.2f' %x, '%d' %y)
```

O/p: 458.54
458.54 10
Result= 458.54 10

To print in a single line

```
print("Beliaghata",end="")
print("Kolkata")
print("RCCIIT","College","Kolkata", sep=",",end=".")
```

O/p: BeliaghataKolkata
RCCIIT,College,Kolkata.

THANK YOU