Introduction To Python

History

Python is a widely-used general-purpose, high-level programming language. It was initially designed by Guido van Rossum in 1991 and developed by Python Software Foundation.

In the late 1980s, Guido Van Rossum began doing its application-based work in December of 1989 at **Centrum Wiskunde & Informatica (CWI)** which is situated in the Netherlands.

The language was finally released in 1991. When it was released, it used a lot fewer codes to express the concepts, when we compare it with Java, C++ & C.

Its main objective is to provide code readability and advanced developer productivity.

Python is Interpreted − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.

Python is Interactive − You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

Python is **Object-Oriented** − Python supports Object-Oriented style or technique of programming that encapsulates code within objects.

Python is a **Beginner's Language** − Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

Features

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++, COM, ActiveX, CORBA, and Java.

Easy to code

- Python is a high-level programming language. Python is very easy to learn the language as compared to other languages like C, C#, Javascript, Java, etc.
- It is very easy to code in the Python language and anybody can learn Python basics in a few hours or days.

Easy to Read

- Python's syntax is really straightforward.
- The code block is defined by the indentations rather than by semicolons or brackets.

Object-Oriented Language

Introduction To Python

- One of the key features of Python is Object-Oriented programming.
- Python supports object-oriented language and concepts of classes, object encapsulation, etc.

High-Level Language

- When we write programs in Python, we do not need to remember the system architecture, nor do we need to manage the memory.

Setting up path

Getting Python

The most up-to-date and current source code, binaries, documentation, news, etc., is available on the official website of Python https://www.python.org/

You can download Python documentation from https://www.python.org/doc/. The documentation is available in HTML, PDF, and PostScript formats.

working with Python

Python Identifiers

A Python identifier is a name used to identify a variable, function, class, module or other object.

An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9).

Python does not allow punctuation characters such as @, $, and % within identifiers.

Python Reserved Words

The following list shows the Python keywords. These are reserved words and you cannot use them as constant or variable or any other identifier names. All the Python keywords contain lowercase letters only.

and	as	assert
break	class	continue
def	del	elif
else	except	False

finally	for	from
global	if	import
in	is	lambda
None	nonlocal	not
or	pass	raise
return	True	try
while	with	yield

Python Lines and Indentation

Python programming provides no braces to indicate blocks of code for class and function definitions or flow control.

The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount. For example −

```
if True:
   print ("True")
else:
   print ("False")
```

However, the following block generates an error .

Basic Syntax

- 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.

Variable and Data Types

Data types are the classification or categorization of data items.

It represents the kind of value that tells what operations can be performed on a particular data.

Introduction To Python

In Python programming, data types are actually classes and variables are instance (object) of these classes.

Following are the standard or built-in data type of Python:

- Numeric
- Sequence Type
- Boolean
- Set
- Dictionary

Numeric

- In Python, numeric data type represent the data which has numeric value.
- Numeric value can be integer, floating number or even complex numbers. These values are defined as int, float and complex class in Python.
    - **Integers** – This value is represented by int class. It contains positive or negative whole numbers (without fraction or decimal). In Python there is no limit to how long an integer value can be.
    - **Float** – This value is represented by float class. It is a real number with floating point representation. It is specified by a decimal point. Optionally, the character e or E followed by a positive or negative integer may be appended to specify scientific notation.
    - **Complex Numbers** – Complex number is represented by complex class. It is specified as *(real part) + (imaginary part)j*. For example – 2+3j

Sequence Type

- In Python, sequence is the ordered collection of similar or different data types.
- Sequences allows to store multiple values in an organized and efficient fashion. There are several sequence types in Python –
    - String
    - List
    - Tuple

1) String

In Python, Strings are arrays of bytes representing Unicode characters. A string is a collection of one or more characters put in a single quote, double-quote or triple quote. In python there is no character data type, a character is a string of length one. It is represented by str class.

2) List

Lists are just like the arrays, declared in other languages which is a ordered collection of data. It is very flexible as the items in a list do not need to be of the same type.

Creating List

Lists in Python can be created by just placing the sequence inside the square brackets[].

Accessing elements of List

In order to access the list items refer to the index number. Use the index operator [ ] to access an item in a list.
In Python, negative sequence indexes represent positions from the end of the array.

Instead of having to compute the offset as in List[len(List)-3], it is enough to just write List[-3].

Negative indexing means beginning from the end, -1 refers to the last item, -2 refers to the second-last item, etc.

3) Tuples

A tuple is a collection of objects which ordered and immutable. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.

Creating a tuple is as simple as putting different comma-separated values. Optionally you can put these comma-separated values between parentheses also. For example −

```
tup1 = ('physics', 'chemistry', 1997, 2000);
tup2 = (1, 2, 3, 4, 5 );
tup3 = "a", "b", "c", "d";
```

The empty tuple is written as two parentheses containing nothing −

```
tup1 = ();
```

To write a tuple containing a single value you have to include a comma, even though there is only one value −

```
tup1 = (50,);
```

Like string indices, tuple indices start at 0, and they can be sliced, concatenated, and so on.

Accessing Values in Tuples

To access values in tuple, use the square brackets for slicing along with the index or indices to obtain value available at that index. For example −

```
#!/usr/bin/python

tup1 = ('physics', 'chemistry', 1997, 2000);
tup2 = (1, 2, 3, 4, 5, 6, 7 );
print "tup1[0]: ", tup1[0];
print "tup2[1:5]: ", tup2[1:5];
```

When the above code is executed, it produces the following result −

Updating Tuples

Tuples are immutable which means you cannot update or change the values of tuple elements. You are able to take portions of existing tuples to create new tuples as the following example demonstrates −

```
#!/usr/bin/python

tup1 = (12, 34.56);
tup2 = ('abc', 'xyz');

# Following action is not valid for tuples
# tup1[0] = 100;

# So let's create a new tuple as follows
tup3 = tup1 + tup2;
print tup3;
```

Delete Tuple Elements

Removing individual tuple elements is not possible. There is, of course, nothing wrong with putting together another tuple with the undesired elements discarded.

To explicitly remove an entire tuple, just use the **del** statement. For example −

```
#!/usr/bin/python

tup = ('physics', 'chemistry', 1997, 2000);
print tup;
del tup;
print "After deleting tup : ";
print tup;
```

Basic Tuples Operations

Tuples respond to the + and * operators much like strings; they mean concatenation and repetition here too, except that the result is a new tuple, not a string.

In fact, tuples respond to all of the general sequence operations we used on strings in the prior chapter −

Python Expression	Results	Description
len((1, 2, 3))	3	Length
(1, 2, 3) + (4, 5, 6)	(1, 2, 3, 4, 5, 6)	Concatenation
('Hi!',) * 4	('Hi!', 'Hi!', 'Hi!', 'Hi!')	Repetition
3 in (1, 2, 3)	True	Membership
for x in (1, 2, 3): print x,	1 2 3	Iteration

Indexing, Slicing, and Matrixes

Because tuples are sequences, indexing and slicing work the same way for tuples as they do for strings. Assuming following input −

```
L = ('spam', 'Spam', 'SPAM!')
```

Python Expression	Results	Description
L[2]	'SPAM!'	Offsets start at zero
L[-2]	'Spam'	Negative: count from the right
L[1:]	['Spam', 'SPAM!']	Slicing fetches sections

No Enclosing Delimiters

Any set of multiple objects, comma-separated, written without identifying symbols, i.e., brackets for lists, parentheses for tuples, etc., default to tuples, as indicated in these short examples

```
#!/usr/bin/python

print 'abc', -4.24e93, 18+6.6j, 'xyz';
x, y = 1, 2;
print "Value of x , y : ", x,y;
```

Built-in Tuple Functions

Python includes the following tuple functions −

Sr.No.	Function with Description
1	cmp(tuple1, tuple2) Compares elements of both tuples.
2	len(tuple) Gives the total length of the tuple.
3	max(tuple) Returns item from the tuple with max value.
4	min(tuple) Returns item from the tuple with min value.
5	tuple(seq) Converts a list into tuple.

Operator

Python operators are the constructs which can manipulate the value of operands.

These are symbols used for the purpose of logical, arithmetic and various other operations.

Types of Python Operators

Python language supports the following types of operators.

- Arithmetic Operators
- Comparison (Relational) Operators

Introduction To Python

- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

Python Arithmetic Operators

Operator	Name	Example
+	Addition	10 + 20 = 30
-	Subtraction	20 – 10 = 10
*	Multiplication	10 * 20 = 200
/	Division	20 / 10 = 2
%	Modulus	22 % 10 = 2
**	Exponent	4**2 = 16
//	Floor Division	9//2 = 4

Python Comparison Operators

Operator	Name	Example
==	Equal	4 == 5 is not true.
!=	Not Equal	4 != 5 is true.
>	Greater Than	4 > 5 is not true.
<	Less Than	4 < 5 is true.
>=	Greater than or Equal to	4 >= 5 is not true.
<=	Less than or Equal to	4 <= 5 is true.

Python Assignment Operators

Operator	Name	Example
=	Assignment Operator	a = 10
+=	Addition Assignment	a += 5 (Same as a = a + 5)
-=	Subtraction Assignment	a -= 5 (Same as a = a - 5)

*=	Multiplication Assignment	a *= 5 (Same as a = a * 5)
/=	Division Assignment	a /= 5 (Same as a = a / 5)
%=	Remainder Assignment	a %= 5 (Same as a = a % 5)
**=	Exponent Assignment	a **= 2 (Same as a = a ** 2)
//=	Floor Division Assignment	a //= 3 (Same as a = a // 3)

Example

Following is an example which shows all the above assignment operations:

```
# Assignment Operator
a = 10

# Addition Assignment
a += 5
print ("a += 5 : ", a)

# Subtraction Assignment
a -= 5
print ("a -= 5 : ", a)

# Multiplication Assignment
a *= 5
print ("a *= 5 : ", a)

# Division Assignment
a /= 5
print ("a /= 5 : ",a)

# Remainder Assignment
a %= 3
```

```
print ("a %= 3 : ", a)


# Exponent Assignment

a **= 2

print ("a **= 2 : ", a)


# Floor Division Assignment

a //= 3

print ("a //= 3 : ", a)
```

Python Bitwise Operators

Bitwise operator works on bits and performs bit by bit operation. Assume if a = 60; and b = 13; Now in the binary format their values will be 0011 1100 and 0000 1101 respectively. Following table lists out the bitwise operators supported by Python language with an example each in those, we use the above two variables (a and b) as operands −

a = 0011 1100

b = 0000 1101

----

a&b = 12 (0000 1100)

a|b = 61 (0011 1101)

a^b = 49 (0011 0001)

~a = -61 (1100 0011)

a << 2 = 240 (1111 0000)

a>>2 = 15 (0000 1111)

There are following Bitwise operators supported by Python language

Operator	Name	Example
&	Binary AND	Sets each bit to 1 if both bits are 1
|	Binary OR	Sets each bit to 1 if one of two bits is 1
^	Binary XOR	Sets each bit to 1 if only one of two bits is 1

~	Binary Ones Complement	Inverts all the bits
<<	Binary Left Shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Binary Right Shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

Python Logical Operators

Operator	Description	Example
and Logical AND	If both the operands are true then condition becomes true.	(a and b) is true.
or Logical OR	If any of the two operands are non-zero then condition becomes true.	(a or b) is true.
not Logical NOT	Used to reverse the logical state of its operand.	Not(a and b) is false.

Python Membership Operators

Operator	Description	Example
in	Evaluates to true if it finds a variable in the specified sequence and false otherwise.	x in y, here in results in a 1 if x is a member of sequence y.
not in	Evaluates to true if it does not finds a variable in the specified sequence and false otherwise.	x not in y, here not in results in a 1 if x is not a member of sequence y.

Python Identity Operators

Operator	Description	Example
is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.	x is y, here **is** results in 1 if id(x) equals id(y).

is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.	x is not y, here **is not** results in 1 if id(x) is not equal to id(y).