Python - Literals

What are Python Literals?

Python literals or constants are the notation for representing a fixed value in source code. In contrast to variables, literals (123, 4.3, "Hello") are static values or you can say constants which do not change throughout the operation of the program or application. For example, in the following assignment statement.

```
x = 10
```

Here 10 is a literal as numeric value representing 10, which is directly stored in memory. However,

```
y = x*2
```

Here, even if the expression evaluates to 20, it is not literally included in source code. You can also declare an int object with built-in int() function. However, this is also an indirect way of instantiation and not with literal.

```
x = int(10)
```

Different Types of Python Literals

Python provides following literals which will be explained this tutorial:

- Integer Literal
- Float Literal
- Complex Literal
- String Literal
- List Literal
- Tuple Literal
- Dictionary Literal

Learn **Python** in-depth with real-world projects through our **Python certification course**. Enroll and become a certified expert to boost your career.

Python Integer Literal

Any representation involving only the digit symbols (0 to 9) creates an object of **int** type. The object so declared may be referred by a variable using an assignment operator.

Integer literals consist three different types of different literal values decimal, octal, and hexadecimal literals.

1. Decimal Literal

Decimal literals represent the signed or unsigned numbers. Digitals from 0 to 9 are used to create a decimal literal value.

Look at the below statement assigning decimal literal to the variable –

```
x = 10
y = -25
z = 0
```

2. Octal Literal

Python allows an integer to be represented as an octal number or a hexadecimal number. A numeric representation with only eight digit symbols (0 to 7) but prefixed by 0o or 0O is an octal number in Python.

Look at the below statement assigning octal literal to the variable –

```
x = 0O34
```

3. Hexadecimal Literal

Similarly, a series of hexadecimal symbols (0 to 9 and a to f), prefixed by 0x or 0X represents an integer in Hexedecimal form in Python.

Look at the below statement assigning hexadecimal literal to the variable –

```
x = 0X1C
```

However, it may be noted that, even if you use octal or hexadecimal literal notation, Python internally treats them as of **int** type.

Example

</> Open Compiler

```
# Using Octal notation
x = 0O34
print ("0O34 in octal is", x, type(x))
# Using Hexadecimal notation
x = 0X1c
print ("0X1c in Hexadecimal is", x, type(x))
```

When you run this code, it will produce the following **output** –

```
0O34 in octal is 28 <class 'int'>
0X1c in Hexadecimal is 28 <class 'int'>
```

Python Float Literal

A floating point number consists of an integral part and a fractional part. Conventionally, a decimal point symbol (.) separates these two parts in a literal representation of a float. For example,

Example of Float Literal

```
x = 25.55
y = 0.05
z = -12.2345
```

For a floating point number which is too large or too small, where number of digits before or after decimal point is more, a scientific notation is used for a compact literal representation. The symbol E or e followed by positive or negative integer, follows after the integer part.

Example of Float Scientific Notation Literal

For example, a number 1.23E05 is equivalent to 123000.00. Similarly, 1.23e-2 is equivalent to 0.0123

</> Open Compiler

```
# Using normal floating point notation
x = 1.23
print ("1.23 in normal float literal is", x, type(x))
```

```
# Using Scientific notation
x = 1.23E5
print ("1.23E5 in scientific notation is", x, type(x))
x = 1.23E-2
print ("1.23E-2 in scientific notation is", x, type(x))
```

Here, you will get the following **output** –

```
1.23 in normal float literal is 1.23 <class 'float'>
1.23E5 in scientific notation is 123000.0 <class 'float''>
1.23E-2 in scientific notation is 0.0123 <class 'float''>
```

Python Complex Literal

A complex number comprises of a real and imaginary component. The imaginary component is any number (integer or floating point) multiplied by square root of "-1"

($\sqrt{-1}$). In literal representation ($\sqrt{-1}$) is representation by "j" or "J". Hence, a literal representation of a complex number takes a form x+yj.

Example of Complex Type Literal

</> Open Compiler

```
#Using literal notation of complex number
x = 2+3j
print ("2+3j complex literal is", x, type(x))
y = 2.5+4.6j
print ("2.5+4.6j complex literal is", x, type(x))
```

This code will produce the following **output** –

```
2+3j complex literal is (2+3j) <class 'complex'>
2.5+4.6j complex literal is (2+3j) <class 'complex'>
```

Python String Literal

A string object is one of the sequence data types in Python. It is an immutable sequence of Unicode code points. Code point is a number corresponding to a character according

to Unicode standard. Strings are objects of Python's built-in class 'str'.

String literals are written by enclosing a sequence of characters in single quotes ('hello'), double quotes ("hello") or triple quotes ('''hello''' or """hello""").

Example of String Literal

</> Open Compiler

```
var1='hello'
print ("'hello' in single quotes is:", var1, type(var1))
var2="hello"
print ('"hello" in double quotes is:', var1, type(var1))
var3='''hello'''
print ("''''hello'''' in triple quotes is:", var1, type(var1))
var4="""hello"""
print ('"""hello""" in triple quotes is:', var1, type(var1))
```

Here, you will get the following **output** –

```
'hello' in single quotes is: hello <class 'str'>
"hello" in double quotes is: hello <class 'str'>
''''hello'''' in triple quotes is: hello <class 'str'>
"""hello""" in triple quotes is: hello <class 'str'>
```

Example of String Literal With Double Quoted Inside String

If it is required to embed double quotes as a part of string, the string itself should be put in single quotes. On the other hand, if single quoted text is to be embedded, string should be written in double quotes.

</> Open Compiler

```
var1='Welcome to "Python Tutorial" from TutorialsPoint'
print (var1)
var2="Welcome to 'Python Tutorial' from TutorialsPoint"
print (var2)
```

It will produce the following **output** –

```
Welcome to "Python Tutorial" from TutorialsPoint
Welcome to 'Python Tutorial' from TutorialsPoint
```

Python List Literal

List object in Python is a collection of objects of other data type. List is an ordered collection of items not necessarily of same type. Individual object in the collection is accessed by index starting with zero.

Literal representation of a list object is done with one or more items which are separated by comma and enclosed in square brackets [].

Example of List Type Literal

</> Open Compiler

```
L1=[1,"Ravi",75.50, True]
print (L1, type(L1))
```

It will produce the following **output** –

```
[1, 'Ravi', 75.5, True] <class 'list'>
```

Python Tuple Literal

Tuple object in Python is a collection of objects of other data type. Tuple is an ordered collection of items not necessarily of same type. Individual object in the collection is accessed by index starting with zero.

Literal representation of a tuple object is done with one or more items which are separated by comma and enclosed in parentheses ().

Example of Tuple Type Literal

</> Open Compiler

```
T1=(1,"Ravi",75.50, True)
print (T1, type(T1))
```

It will produce the following **output** –

```
[1, 'Ravi', 75.5, True] <class tuple>
```

Example of Tuple Type Literal Without Parenthesis

Default delimiter for Python sequence is parentheses, which means a comma separated sequence without parentheses also amounts to declaration of a tuple.

</> Open Compiler

```
T1=1,"Ravi",75.50, True
print (T1, type(T1))
```

Here too, you will get the same **output** –

```
[1, 'Ravi', 75.5, True] <class tuple>
```

Python Dictionary Literal

Like list or tuple, dictionary is also a collection data type. However, it is not a sequence. It is an unordered collection of items, each of which is a key-value pair. Value is bound to key by the ":" symbol. One or more key:value pairs separated by comma are put inside curly brackets to form a dictionary object.

Example of Dictionary Type Literal

</> Open Compiler

```
capitals={"USA":"New York", "France":"Paris", "Japan":"Tokyo",
"India":"New Delhi"}
numbers={1:"one", 2:"Two", 3:"three",4:"four"}
points={"p1":(10,10), "p2":(20,20)}

print (capitals, type(capitals))
print (numbers, type(numbers))
print (points, type(points))
```

Key should be an immutable object. Number, string or tuple can be used as key. Key cannot appear more than once in one collection. If a key appears more than once, only the last one will be retained. Values can be of any data type. One value can be assigned to more than one keys. For example,

```
staff={"Krishna":"Officer", "Rajesh":"Manager", "Ragini":"officer",
"Anil":"Clerk", "Kavita":"Manager"}
```