Python Variables and Literals

In the previous tutorial you learned about Python comments. Now, let's learn about variables and literals in Python.

Python Variables

In programming, a variable is a container (storage area) to hold data. For example,

```
number = 10
```

Here, *number* is a variable storing the value **10**.

Assigning values to Variables in Python

As we can see from the above example, we use the assignment operator = to assign a value to a variable.

```
# assign value to site_name variable
site_name = 'programiz.pro'

print(site_name)

# Output: programiz.pro
```

Output

```
programiz.pro
```

In the above example, we assigned the value `programiz.pro` to the *site_name* variable. Then, we printed out the value assigned to *site_name*

Note: Python is a type-inferred language, so you don't have to explicitly define the variable type. It automatically knows that `programiz.pro` is a string and declares the `site_name` variable as a string.

Changing the Value of a Variable in Python

```
site_name = 'programiz.pro'
print(site_name)

# assigning a new value to site_name
site_name = 'apple.com'

print(site_name)
```

Output

```
programiz.pro
apple.com
```

Here, the value of *site_name* is changed from `'programiz.pro'` to `'apple.com'`.

Example: Assigning multiple values to multiple variables

```
a, b, c = 5, 3.2, 'Hello'

print (a)  # prints 5
print (b)  # prints 3.2
print (c)  # prints Hello
```

If we want to assign the same value to multiple variables at once, we can do this as:

```
site1 = site2  = 'programiz.com'

print (x)  # prints programiz.com
print (y)  # prints programiz.com
```

Here, we have assigned the same string value `'programiz.com'` to both the variables *site1* and *site2*.

Rules for Naming Python Variables

1. Constant and variable names should have a combination of letters in lowercase (a to z) or uppercase (**A to Z**) or digits (**0 to 9**) or an

underscore (_). For example:

```
snake_case
MACRO_CASE
camelCase
CapWords
```

2. Create a name that makes sense. For example, *vowel* makes more sense than *v*.

3. If you want to create a variable name having two words, use underscore to separate them. For example:

```
my_name
current_salary
```

5. Python is case-sensitive. So *num* and *Num* are different variables. For example,

```
var num = 5
var Num = 55
print(num) # 5
print(Num) # 55
```

6. Avoid using keywords like if, True, class, etc. as variable names.

Python Literals

Literals are representations of fixed values in a program. They can be numbers, characters, or strings, etc. For example, `'Hello, World!'`, `12`, `23.0`, `'C'`, etc.

Literals are often used to assign values to variables or constants. For example,

```
site_name = 'programiz.com'
```

In the above expression, *site_name* is a variable, and `'programiz.com'` is a literal.

There are different types of literals in Python. Let's discuss some of the commonly used types in detail.

Python Numeric Literals

Numeric Literals are immutable (unchangeable). Numeric literals can belong to **3** different numerical types: `Integer`, `Float`, and `Complex`.

1. Integer Literals

Integer literals are numbers without decimal parts. It also consists of negative numbers. For example, `5`, `-11`, `0`, `12`, etc.

2. Floating-Point Literals

Floating-point literals are numbers that contain decimal parts.

Just like integers, floating-point numbers can also be both positive and negative. For example, `2.5`, `6.76`, `0.0`, `-9.45`, etc.

3. Complex Literals

Complex literals are numbers that represent complex numbers.

Here, numerals are in the form `a + bj`, where `a` is real and `b` is imaginary. For example, `6+9j`, `2+3j`.

Python String Literals

In Python, texts wrapped inside quotation marks are called **string literals.**.

```
"This is a string."
```

We can also use single quotes to create strings.

```
'This is also a string.'
```

More on Python Literals

Python Boolean Literals

There are two boolean literals: `True` and `False`.

For example,

```
is_pass = True
print(is_pass)

# Output: True
```

Here, `True` is a boolean literal assigned to *is_pass*.

Character Literals in Python

Character literals are unicode characters enclosed in a quote. For example,

```
some_character = 'S'
```

Here, `S` is a character literal assigned to *some_character*.

Special Literal in Python

Python contains one special literal `None`. We use it to specify a null variable. For example,

```
value = None

print(value)

# Output: None
```

Here, we get `None` as an output as the *value* variable has no value assigned to it.

Collection Literals

Let's see examples of four different collection literals. List, Tuple, Dict, and Set literals.

```
# list literal
fruits = ["apple", "mango", "orange"]
print(fruits)

# tuple literal
numbers = (1, 2, 3)
print(numbers)

# dictionary literal
alphabets = {'a':'apple', 'b':'ball', 'c':'cat'}
print(alphabets)

# set literal
vowels = {'a', 'e', 'i' , 'o', 'u'}
print(vowels)
```

Output

```
['apple', 'mango', 'orange']
(1, 2, 3)
{'a': 'apple', 'b': 'ball', 'c': 'cat'}
{'e', 'a', 'o', 'i', 'u'}
```

In the above example, we created a list of *fruits*, a tuple of *numbers*, a dictionary of *alphabets* having values with keys designated to each value and a set of *vowels*.

To learn more about literal collections, refer to Python Data Types.

Table of Contents

Video: Python Variables and print()

Previous Tutorial:
Python Comments
Next Tutorial:
Python Type Conversion

Share on:

Did you find this article helpful?