Python Variables

Last Updated : 14 Jan, 2026

In Python, variables are used to store data that can be referenced and manipulated during program execution. A variable is essentially a name that is assigned to a value.

- Unlike Java and many other languages, Python variables do not require explicit declaration of type.
- The type of the variable is inferred based on the value assigned.

```
x = 5
name = "Samantha"
print(x)
print(name)
```

Output

```
5
Samantha
```

Rules for Naming Variables

To use variables effectively, we must follow Python's naming rules:

1. Variable names can only contain letters, digits and underscores (_).
2. A variable name cannot start with a digit.
3. Variable names are case-sensitive like myVar and myvar are different.
4. Avoid using Python keywords like if, else, for as variable names.

Below listed variable names are valid:

```
age = 21
_colour = "lilac"
total_score = 90
```

Below listed variables names are invalid:

```
1name = "Error"  # Starts with a digit
class = 10       # 'class' is a reserved keyword
user-name = "Doe"  # Contains a hyphen
```

Assigning Values to Variables

Basic Assignment: Variables in Python are assigned values using the = operator.

```
x = 5
y = 3.14
z = "Hi"
```

Dynamic Typing: Python variables are dynamically typed, meaning the same variable can hold different types of values during execution.

```
x = 10
x = "Now a string"
```

Multiple Assignments

Assigning Same Value: Python allows assigning the same value to multiple variables in a single line, which can be useful for initializing variables with the same value.

```
a = b = c = 100
print(a, b, c)
```

Output

```
100 100 100
```

Assigning Different Values: We can assign different values to multiple variables simultaneously, making the code concise and easier to read.

```
x, y, z = 1, 2.5, "Python"
print(x, y, z)
```

Output

```
1 2.5 Python
```

Type Casting a Variable

Type casting refers to the process of converting the value of one data type into another. Python provides several built-in functions to facilitate casting, including int(), float() and str() among others. Basic casting functions are:

- **int()**: Converts compatible values to an integer.
- **float()**: Transforms values into floating-point numbers.
- **str()**: Converts any data type into a string.

```
s = "10"
n = int(s)

cnt = 5
f = float(cnt)

age = 25
s2 = str(age)

print(n)
print(f)
print(s2)
```

Output

```
10
5.0
25
```

Type of Variable

In Python, we can determine the type of a variable using the type() function. This built-in function returns the type of the object passed to it.

```
n = 42
f = 3.14
s = "Hello, World!"
li = [1, 2, 3]
d = {'key': 'value'}
bool = True

print(type(n))
print(type(f))
print(type(s))
print(type(li))
print(type(d))
print(type(bool))
```

Output

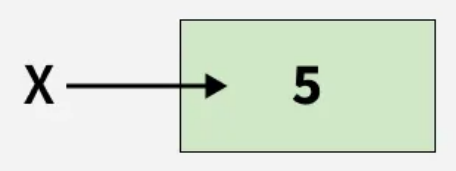
```
<class 'int'>
<class 'float'>
<class 'str'>
<class 'list'>
<class 'dict'>
<class 'bool'>
```

Concept of Object Reference

Let us assign a variable x to value 5.

x = 5

When x = 5 is executed, Python creates an object to represent the value 5 and makes x reference this object.

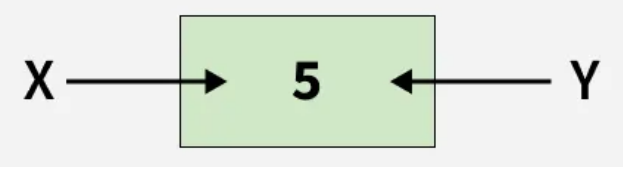


object reference

Now, let's assign another variable y to the variable x.

y = x

This statement creates y and references the same object as x, not x itself. This is called a Shared Reference, where multiple variables reference the same object.

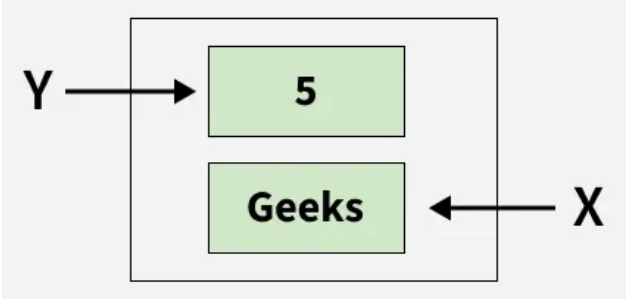


shared reference

Now, if we write

x = 'Geeks'

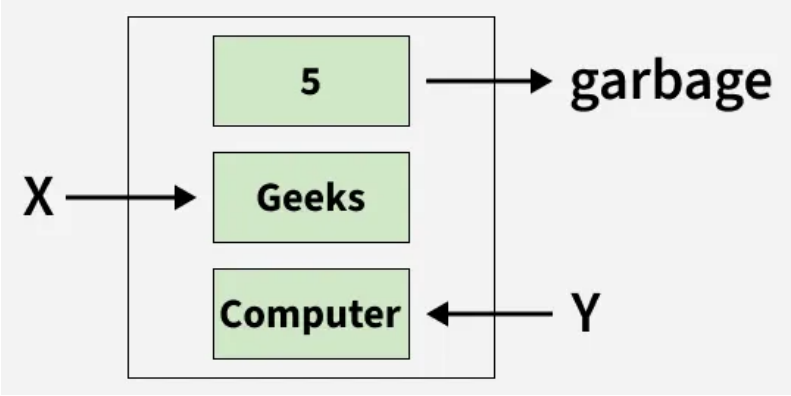
Python creates a new object for the value "Geeks" and makes x reference this new object.



shared reference example

The variable y remains unchanged, still referencing the original object 5. Now, If we assign a new value to y:

y = "Computer"



shared reference example

- Python creates yet another object for "Computer" and updates y to reference it.
- The original object 5 no longer has any references and becomes eligible for garbage collection.
- Python variables hold references to objects, not the actual objects themselves.
- Reassigning a variable does not affect other variables referencing the same object unless explicitly updated.

Deleting a Variable

We can remove a variable from the namespace using the del keyword. This deletes the variable and frees up the memory it was using.

```
x = 10
del x
print(x)
```

Output

ERROR!
Traceback (most recent call last):
File "<main.py>", line 3, in <module>
NameError: name 'x' is not defined

Explanation:

- del x removes the variable x from memory.
- After deletion, trying to access the variable x results in a NameError indicating that the variable no longer exists.

Practical Examples

1. **Swapping Two Variables:** Using multiple assignments, we can swap the values of two variables without needing a temporary variable.

```
a, b = 5, 10
a, b = b, a
print(a, b)
```

Output

```
10 5
```

2. **Counting Characters in a String:** Assign the results of multiple operations on a string to variables in one line.

```
word = "Python"
length = len(word)
print("Length of the word:", length)
```

Output

```
Length of the word: 6
```

Related Posts

- Python Variable Quiz
- Local and Global Scope
- Assign function to a Variable in Python
- Insert a Variable into a String in Python
- Type Casting in Python

Recommended Problems

- Type Conversion
- TypeCast And Double It
- Swap The Numbers
- Sum of N Numbers
- Int Str

Suggested Quiz 8 Questions

What is a variable in Python?

A A function that performs calculations.

B A named location used to store data in memory.

C A loop that iterates over a sequence.

D A module imported into a Python script.

Login to View Explanation 1/8 < Previous Next >

Comment S shruthi... + Follow 90

Article Tags: Python python-basics