

Variables in Python

by John Sturtz   [basics](#) [python](#)
[Tweet](#) [Share](#) [Email](#)

Table of Contents

- [Variable Assignment](#)
- [Variable Types in Python](#)
- [Object References](#)
- [Object Identity](#)
- [Variable Names](#)
- [Reserved Words \(Keywords\)](#)
- [Conclusion](#)



Master **Real-World Python Skills**
With a **Community of Experts**

Level Up With Unlimited Access to Our Vast Library
of Python Tutorials and Video Lessons

Watch Now »

In the previous tutorial on [Basic Data Types in Python](#), you saw how values of various Python data types can be created. But so far, all the values shown have been literal or constant values:


Python>>>

```
>>> print(5.3)
5.3
```

If you’re writing more complex code, your program will need data that can change as program execution proceeds.

Here’s what you’ll learn in this tutorial: You will learn how every item of data in a Python program can be described by the abstract term **object**, and you’ll learn how to manipulate objects using symbolic names called **variables**.

Free PDF Download: [Python 3 Cheat Sheet](#)

 **Take the Quiz:** Test your knowledge with our interactive “Python Variables” quiz. Upon completion you will receive a score so you can track your learning progress c **Improve Your Python**

Take the Quiz »

Variable Assignment

Think of a variable as a name attached to a particular object. In Python, variables need not be declared or defined in advance, as is the case in many other programming languages. To create a variable, you just assign it a value and then start using it. Assignment is done with a single equals sign (=):

Python

```
>>> n = 300
```

This is read or interpreted as “n is assign expression, and its value will be substitu

Python

```
>>> print(n)
300
```

Just as a literal value can be displayed d print(), so can a variable:

Python

```
>>> n
300
```

Later, if you change the value of n and use it again, the new value will be substituted instead:

Python

```
>>> n = 1000
>>> print(n)
1000
>>> n
1000
```

Python also allows chained assignment, which makes it possible to assign the same value to several variables simultaneously:

Python

```
>>> a = b = c = 300
>>> print(a, b, c)
300 300 300
```

The chained assignment above assigns 300 to the variables a, b, and c simultaneously.

Variable Types in Python

In many programming languages, variables are statically typed. That means a variable is initially declared to have a specific data type, and any value assigned to it during its lifetime must always have that type.

Variables in Python are not subject to this restriction. In Python, a variable may be assigned a value of one type and then later re-assigned a value of a different type:

Python

```
>>>
```

Improve Your Python

...with a fresh  **Python Trick**  code snippet every couple of days:

Email Address

☐ Receive the Real Python newsletter and get notified about new tutorials we publish on the site, as well as occasional special offers.

Send Python Tricks »

```
>>> var = 23.5
>>> print(var)
23.5

>>> var = "Now I'm a string"
>>> print(var)
Now I'm a string
```

Object References

What is actually happening when you m the answer differs somewhat from what

Python is a highly [object-oriented langu](#) specific type or class. (This point will be

Consider this code:

```
Python

>>> print(300)
300
```

When presented with the statement pri

- Creates an integer object
- Gives it the value 300
- Displays it to the console

You can see that an integer object is created using the built-in type() function:

```
Python                                     >>>

>>> type(300)
<class 'int'>
```

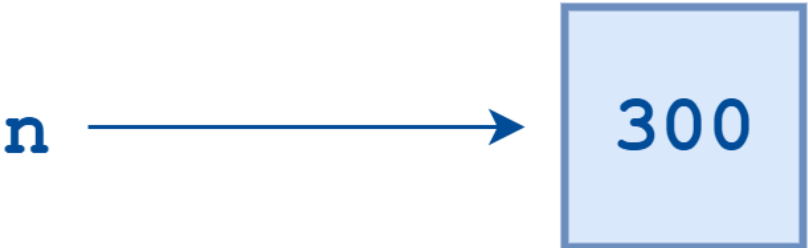
A Python variable is a symbolic name that is a reference or pointer to an object. Once an object is assigned to a variable, you can refer to the object by that name. But the data itself is still contained within the object.

For example:

```
Python                                     >>>

>>> n = 300
```

This assignment creates an integer object with the value 300 and assigns the variable n to point to that object.



Variable Assignment

The following code verifies that n points to an integer object:

```
Python                                     >>>

>>> print(n)
300
>>> type(n)
<class 'int'>
```

Now consider the following statement:

Improve Your Python

Improve Your Python

...with a fresh **Python Trick** code snippet every couple of days:

Email Address

☐ Receive the Real Python newsletter and get notified about new tutorials we publish on the site, as well as occasional special offers.

Send Python Tricks »

Python

>>> m = n

What happens when it is executed? Python does not create another object. It simply creates a new symbolic name or reference, `m`, which points to the same object that `n` points to.



Next, suppose you do this:

Python

>>> m = 400

Now Python creates a new integer objec



```
1# How to merge two dicts
2# in Python 3.5+
3
4>>> x = {'a': 1, 'b': 2}
5>>> y = {'b': 3, 'c': 4}
6
7>>> z = {**x, **y}
8
9>>> z
10{'c': 4, 'a': 1, 'b': 3}
```

Improve Your Python

...with a fresh  **Python Trick**  code snippet every couple of days:

Email Address

☐ Receive the Real Python newsletter and get notified about new tutorials we publish on the site, as well as occasional special offers.

[Send Python Tricks »](#)

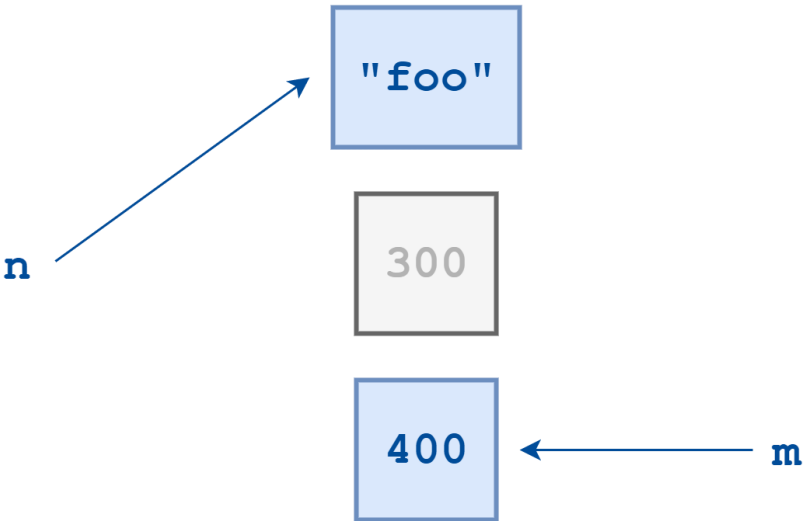
References to Separate Objects

Lastly, suppose this statement is executed next:

Python

>>> n = "foo"

Now Python creates a string object with the value "foo" and makes `n` reference that.



Orphaned Object

There is no longer any reference to the integer object `300`. It is orphaned, and there is no way to access it.

Tutorials in this series will occasionally refer to the lifetime of an object. An object’s life begins when it is created, at which time at least one reference to it is created. During an object’s lifetime, additional references to it may be created, as you saw above, and references to it may be deleted as well. An object stays alive, as it were, so long as there is at least one reference to it.

When the number of references to an object drops to zero, it is no longer accessible. At that point, its lifetime is over. Python will eventually notice that it is inaccessible and reclaim the allocated memory so it can be used for something else. In computer lingo, this process is referred to as [garbage collection](#).

Object Identity

Improve Your Python

In Python, every object that is created is given a number that uniquely identifies it. It is guaranteed that no two objects will have the same identifier during any period in which their lifetimes overlap. Once an object’s reference count drops to zero and it is garbage collected, as happened to the 300 object above, then its identifying number becomes available and may be used again.

The built-in Python function `id()` returns an object’s integer identifier. Using the `id()` function, you can verify that two variables indeed point to the same object:



Python

>>> n = 300
>>> m = n
>>> id(n)
60127840
>>> id(m)
60127840

>>> m = 400
>>> id(m)
60127872

```
1# How to merge two dicts
2# in Python 3.5+
3
4>>> x = {'a': 1, 'b': 2}
5>>> y = {'b': 3, 'c': 4}
6
7>>> z = {**x, **y}
8
9>>> z
10{'c': 4, 'a': 1, 'b': 3}
```

Improve Your Python

...with a fresh  Python Trick  code snippet every couple of days:

Email Address

☐ Receive the Real Python newsletter and get notified about new tutorials we publish on the site, as well as occasional special offers.

Send Python Tricks »

After the assignment `m = n`, `m` and `n` both the same number. Once `m` is reassigned t

Deep Dive: Caching Sma

From what you now know about vari won’t surprise you:

Python

>>> m = 300
>>> n = 300
>>> id(m)
60062304
>>> id(n)
60062896

With the statement `m = 300`, Python creates an integer object with the value 300 and sets `m` as a reference to it. `n` is then similarly assigned to an integer object with value 300—but not the same object. Thus, they have different identities, which you can verify from the values returned by `id()`.

But consider this:

Python

>>> m = 30
>>> n = 30
>>> id(m)
1405569120
>>> id(n)
1405569120

Here, `m` and `n` are separately assigned to integer objects having value 30. But in this case, `id(m)` and `id(n)` are identical!

For purposes of optimization, the interpreter creates objects for the integers in the range `[-5, 256]` at startup, and then reuses them during program execution. Thus, when you assign separate variables to an integer value in this range, they will actually reference the same object.

Variable Names

The examples you have seen so far have used short, terse variable names like `m` and `n`. But variable names can be more verbose. In fact, it is usually beneficial if they are because it makes the purpose of the variable more evident at first glance.

Improve Your Python

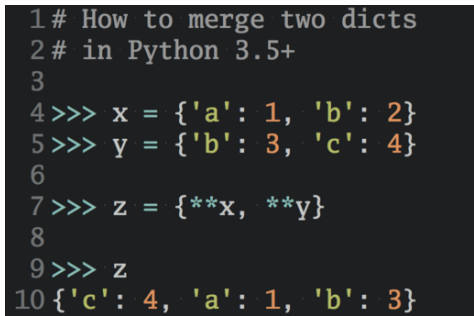
Officially, variable names in Python can be any length and can consist of uppercase and lowercase letters (A-Z, a-z), digits (0-9), and the underscore character (`_`). An additional restriction is that, although a variable name can contain digits, the first character of a variable name cannot be a digit.

Note: One of the additions to Python 3 was full Unicode support, which allows for Unicode characters in a variable name as well. You will learn about Unicode in greater depth in a future tutorial.

For example, all of the following are valid variable names:

Python

```
>>> name = "Bob"
>>> Age = 54
>>> has_w2 = True
>>> print(name, Age, has_w2)
Bob 54 True
```



Improve Your Python

...with a fresh  **Python Trick**  code snippet every couple of days:

Email Address

☐ Receive the Real Python newsletter and get notified about new tutorials we publish on the site, as well as occasional special offers.

[Send Python Tricks »](#)

But this one is not, because a variable name

Python

```
>>> 1099_filed = False
SyntaxError: invalid token
```

Note that case is significant. Lowercase and uppercase are significant as well. Each of the following

Python

```
>>> age = 1
>>> Age = 2
>>> aGe = 3
>>> AGE = 4
>>> a_g_e = 5
>>> _age = 6
>>> age_ = 7
>>> _AGE_ = 8

>>> print(age, Age, aGe, AGE, a_g_e, _age, age_, _AGE_)
1 2 3 4 5 6 7 8
```

There is nothing stopping you from creating two different variables in the same program called `age` and `Age`, or for that matter `agE`. But it is probably ill-advised. It would certainly be likely to confuse anyone trying to read your code, and even you yourself, after you'd been away from it awhile.

It is worthwhile to give a variable a name that is descriptive enough to make clear what it is being used for. For example, suppose you are tallying the number of people who have graduated college. You could conceivably choose any of the following:

Python

```
>>> numberofcollegegraduates = 2500
>>> NUMBEROFCOLLEGEGRADUATES = 2500
>>> numberOfCollegeGraduates = 2500
>>> NumberOfCollegeGraduates = 2500
>>> number_of_college_graduates = 2500

>>> print(numberofcollegegraduates, NUMBEROFCOLLEGEGRADUATES,
...       numberOfCollegeGraduates, NumberOfCollegeGraduates,
...       number_of_college_graduates)
2500 2500 2500 2500 2500
```

All of them are probably better choices than `n`, or `ncg`, or the like. At least you can tell from the name what the value of the variable is supposed to represent.

On the other hand, they aren’t all necessarily equally legible. As with many things, it is a matter of personal preference, but most people would find the first two examples, where the letters are all shoved together, to be harder to read, particularly the one in all capital letters. The most commonly used methods of constructing a multi-word variable name are the last three examples:

- **Camel Case:** Second and subsequent words are capitalized, to make word boundaries easier to see. (Presumably, it struck someone at some point that the capital letters strewn throughout the variable name vaguely resemble camel humps.)
 - Example: `numberOfCollegeGraduates`
- **Pascal Case:** Identical to Camel Case
 - Example: `NumberOfCollegeGra`
- **Snake Case:** Words are separated by underscores
 - Example: `number_of_college_`

Programmers debate hotly, with surprise, of them. Use whichever of the three is most

You will see later that variables aren’t the only things that can be named with Python modules, and so on. The rules that apply to naming variables are also given to program objects.

The [Style Guide for Python Code](#), also known as PEP 8, provides guidelines for names of different object types. PEP 8

- Snake Case should be used for functions and variable names.
- Pascal Case should be used for class names. (PEP 8 refers to this as the “CapWords” convention.)

Reserved Words (Keywords)

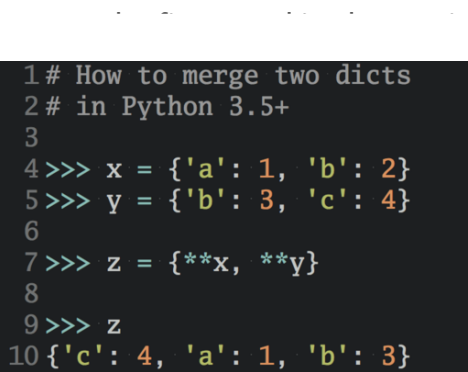
There is one more restriction on identifier names. The Python language reserves a small set of keywords that designate special language functionality. No object can have the same name as a reserved word.

In Python 3.6, there are 33 reserved keywords:

Python Keywords			
False	def	if	raise
None	del	import	return
True	elif	in	try
and	else	is	while
as	except	lambda	with
assert	finally	nonlocal	yield
break	for	not	
class	from	or	
continue	global	pass	

You can see this list any time by typing `help("keywords")` to the Python interpreter. Reserved words are case-sensitive and must be used exactly as shown. They are all entirely lowercase, except for `False`, `None`, and `True`.

Trying to create a variable with the same name as any reserved word results in an error.



Improve Your Python

...with a fresh  **Python Trick** 
code snippet every couple of days:

Email Address

☐ Receive the Real Python newsletter and get notified about new tutorials we publish on the site, as well as occasional special offers.

[Send Python Tricks »](#)

Python

>>>


>>> for = 3
SyntaxError: invalid syntax

Conclusion

This tutorial covered the basics of Python **variables**, including object references and identity, and naming of Python identifiers.

You now have a good understanding of s
objects of those types.

Next, you will see how to combine data c

 **Take the Quiz:** Test your knowled
receive a score so you can track your

```
1# How to merge two dicts  
2# in Python 3.5+  
3  
4>>> x = {'a': 1, 'b': 2}  
5>>> y = {'b': 3, 'c': 4}  
6  
7>>> z = {**x, **y}  
8  
9>>> z  
10{'c': 4, 'a': 1, 'b': 3}
```

Improve Your Python



...with a fresh  **Python Trick** 
code snippet every couple of days:

Email Address

☐ Receive the Real Python newsletter and get notified about new tutorials we publish on the site, as well as occasional special offers.

[« Basic Data Types in Python](#)

[Send Python Tricks »](#) [s in](#)
[n »](#)

 Python Tricks 


Get a short & sweet **Python Trick** delivered to your inbox every couple of days. No spam ever. Unsubscribe any time. Curated by the Real Python team.

Email Address

Send Me Python Tricks »

```
1# How to merge two dicts  
2# in Python 3.5+  
3  
4>>> x = {'a': 1, 'b': 2}  
5>>> y = {'b': 3, 'c': 4}  
6  
7>>> z = {**x, **y}  
8  
9>>> z  
10{'c': 4, 'a': 1, 'b': 3}
```

About **John Sturtz**

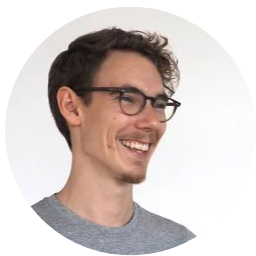


John is an avid Pythonista and a member of the Real Python tutorial team.

[» More about John](#)

Each tutorial at Real Python is created by a team of developers. The team who worked on this tutorial are:

Improve Your Python



[Dan](#)



[Joanna](#)



Master Python 3 and write more Pythonic code with our in-depth



```
1# How to merge two dicts
2# in Python 3.5+
3
4>>> x = {'a': 1, 'b': 2}
5>>> y = {'b': 3, 'c': 4}
6
7>>> z = {**x, **y}
8
9>>> z
10{'c': 4, 'a': 1, 'b': 3}
```

Improve Your Python



...with a fresh **Python Trick** code snippet every couple of days:

☐ Receive the Real Python newsletter and get notified about new tutorials we publish on the site, as well as occasional special offers.

[Send Python Tricks »](#)

What Do You Think?

Real Python Comment Policy: Th from or helping out other readers- Complaints and insults generally won't make the cut here.

Keep Learning

Related Tutorial Categories: [basics](#) [python](#)

— FREE Email Series —



```
1# How to merge two dicts
2# in Python 3.5+
3
4>>> x = {'a': 1, 'b': 2}
5>>> y = {'b': 3, 'c': 4}
6
7>>> z = {**x, **y}
8
9>>> z
10{'c': 4, 'a': 1, 'b': 3}
```


[Get Python Tricks »](#)

No spam. Unsubscribe any time.

[All Tutorial Topics](#)

[Improve Your Python](#)

- [advanced](#)
- [api](#)
- [basics](#)
- [best-practices](#)
- [community](#)
- [databases](#)
- [data-science](#)
- [devops](#)
- [django](#)
- [docker](#)
- [flask](#)
- [front-end](#)
- [intermediate](#)
- [machine-learning](#)
- [python](#)
- [testing](#)
- [tools](#)
- [web-dev](#)
- [web-scraping](#)





Master Python With a Level Up

Our Video Courses

```
1# How to merge two dicts
2# in Python 3.5+
3
4>>> x = {'a': 1, 'b': 2}
5>>> y = {'b': 3, 'c': 4}
6
7>>> z = {**x, **y}
8
9>>> z
10{'c': 4, 'a': 1, 'b': 3}
```

Improve Your Python

...with a fresh  **Python Trick** 

code snippet every couple of days:

Email Address

☐ Receive the Real Python newsletter and get notified about new tutorials we publish on the site, as well as occasional special offers.

[Send Python Tricks »](#)

Table of Contents

- [Variable Assignment](#)
- [Variable Types in Python](#)
- [Object References](#)
- [Object Identity](#)
- [Variable Names](#)
- [Reserved Words \(Keywords\)](#)
- [Conclusion](#)

-  [Tweet](#)
-  [Share](#)
-  [Email](#)



High Quality Python Video Courses

[Watch Now »](#)

```
1 # How to merge two dicts
2 # in Python 3.5+
3
4 >>> x = {'a': 1, 'b': 2}
5 >>> y = {'b': 3, 'c': 4}
6
7 >>> z = {**x, **y}
8
9 >>> z
10 {'c': 4, 'a': 1, 'b': 3}
```

Improve Your Python ×

...with a fresh  **Python Trick** 
code snippet every couple of days:

Email Address

☐ Receive the Real Python newsletter and get notified about new tutorials we publish on the site, as well as occasional special offers.

[Send Python Tricks »](#)