PREV
Lesson 3. Introducing Python: a programming lan

NEXT
Lesson 5. Object types and statements of code

Lesson 4. Variables and expressions: giving names and values to things

After reading lesson 4, you'll be able to

- Write code that creates Python objects
- Write code that assigns objects to variables

In your everyday life, you encounter many physical objects, or *things*. Each of these things has a name. They have names because it's much easier to refer to them using a name rather than a description.

Using names is a great help when you're always manipulating things, or objects. Some things are simple, such as the number 9. Some are more complicated, such as a dictionary. I can give the name *Nana* to the number 9, and the name *Bill* to my dictionary. You can give things (almost) any name you want. You can even give names to combinations of things. For example, if I glue a banana to my laptop cover to create a new thing, I can name that new trendy creation Banalaptop. Individual things can be named as well; if I have two apples, I can name one Allie and the other one Ollie.

After you name things, you can refer to them later without any confusion. The benefit of using names is that you don't have to re-create (in programming, *recalculate*) values. When you name a thing, you inherently remember every detail about it.

Consider this

Scan the room you're in right now for a few things. Then take these steps:

1. Write the items down. (I saw my phone, a chair, a carpet, papers, and a water bottle.)
2. Write a sentence using some or all of these objects. You can use an object more than once. (My water bottle spilled on my phone and papers, and now my phone is broken and my papers are ruined.)
3. Write a description of each object without using its name.
4. Now rewrite the sentence you came up with using only the descriptions.
5. Is the sentence you wrote easier to read using the item names or descriptions?

Answers:

1. A phone, papers, and a water bottle.
2. My water bottle spilled on my phone and papers, and now my phone is broken and my papers are ruined.
3. Descriptions:
   - *Water bottle*—Thing that holds clear liquid
   - *Phone*—Rectangular device I use to call/text/watch cat videos
   - *Papers*—Stack of thin, white, flimsy things with black-and-white text on them
4. A thing that holds clear liquid spilled on a rectangular device I use to call/text/watch cat videos and on a stack of thin, white, flimsy things with black-and-white text on them, and now my rectangular device is broken, and my things with black-and-white text are ruined.
5. The sentence is easier to read with the item names, not the descriptions.

Everything you use has a name, which makes it easier to reference in conversation. Writing a computer program is like writing a detailed description of the events you want to happen and the things involved. In programming, you reference things by using variables, which are discussed in the context of programming in section 4.2.

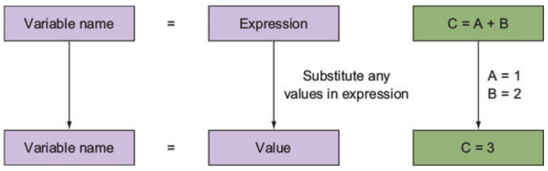
4.1.1. Math vs. programming

When you hear the word *variable*, it may remind you of math class, where you did calculations with equations and were asked to "solve for *x*." Programming also uses variables, but in a different way.

In math, lines of equations state an equivalence. For example, "*x* = 1" stands for "*x* is equivalent to 1," and "2 * *x* = 3 * y" stands for "2 times x is equivalent to 3 times *y*."

In programming, lines of code with an equal sign stand for an *assignment*. Figure 4.1 shows an assignment in Python.

Figure 4.1. Assignment to a name in Python. Any expression on the right side gets converted to a single value and given a name.



You use the equal sign to assign variables to values. For example, `a = 1` or `c = a + b`. The thing to the right of the equal sign is an *expression* with a *value*.

Definition

An expression is a line of code that can be reduced to a value.

To get the value, you substitute the values for all other known variables in the expression and do the calculation. For example, if `a = 1` and `b = 2`, then `c = a + b = 1 + 2 = 3`. In programming, the only thing you're allowed to have to the left of the equal sign is the name of a variable.

4.1.2. What the computer can and can't do

An important point is worth coming back to: a computer needs to be told what to do. A computer can't spontaneously solve an equation on its own. If you tell the computer that `a = 2`, `b = 2`, and that `a + x = b`, it doesn't know what to do with this information or how to solve for `x` on its own. The line `a + x = b` doesn't tell the computer how to calculate anything; it just states an equivalence.

The computer needs to be told a recipe for solving something. Recall that when you're following a recipe for baking bread, you need to know the steps. You, as the programmer, have to come up with the recipe and tell the computer what to do. To come up with the recipe, you need to go off-computer and on-paper and do the calculation on your own. Then you can tell the computer what steps it needs to calculate a value.

Quick check 4.1

Decide whether the computer is allowed to do the following assignments. Assume that every *thing* on the right side of the equal sign has a value:

1

```
3 + 3 = 4
```

2

```
stuff + things = junk
```

3

```
stack = 1000 * papers + 40 * envelopes
```

4

```
seasons = spring + summer + fall + winter
```

With that bit of intuition for how variables work in programming, you can now dive in and start learning about how variables work.

4.2.1. Objects are things that can be manipulated

In the previous section, we talked about things. In Python, everything is an object. This means that every *thing* that you can create in Python has the following:

- A type
- A set of operations

The *type* of an object tells you the data/values/attributes/properties associated with it. The *operations* are commands that you can tell the object to do; these commands might work only on the object itself, or they might be ways that the object can interact with other objects.

Quick check 4.2

Q1:

For the following items, write some attributes (describe their color, size, and so forth) and some operations (what it can do, how it can interact with something else, and so forth):

1. Phone
2. Dog
3. Mirror
4. Credit card

4.2.2. Objects have names

Every *thing* that you create in a program can be given a name so you can refer to it later. The names are *variables* and are used to refer to objects.

Definition

A variable is used to bind a name to an object. A variable name refers to a particular object.

For example:

- If `a = 1`, then the object named `a` has the value `1`, and you can do mathematical operations with it.
- If `greeting = "hello"`, then the object named `greeting` has a value of `"hello"` (a sequence of characters). Operations you can do on this object include “tell me how many characters it has” or “tell me if it has the letter *a* in it” or “tell me at which position the first *e* occurs.”

In both of these examples, the item to the left of the equal sign is a variable name that you can use to refer to an object, and the thing to the right of the equal sign is the object itself, which has a value and some operations you can do on it. In Python, you bind a variable name to an object.

The object to the right of the equal sign doesn’t have to be only one object. It can be a calculation that can be simplified to give a value. With that final value, you get an object. For example, `a = 1 + 2` is a calculation on two objects (`1` and `2`) that can be simplified to one object with a value of `3`.

4.2.3. What object names are allowed?

You write code with variable names that make the code readable by other people. Many programming languages, Python included, have restrictions on the names you can use for variables:

- Must begin with a letter (`a` to `z` or `A` to `Z`) or an underscore (_).
- Other characters in the variable name can be letters, numbers, or an underscore.
- Names are case-sensitive.
- Names can be any length.

Thinking like a programmer

If you want, you can have a variable name that is 1,000,000,000,000,000 characters long. But don’t! It makes your code unreadable. Limit lines of code to at most 80 characters and try to make your names as concise as possible while maintaining readability.

Quick check 4.3

Are the following variable names allowed?

1

`A`

2

`a-number`

3

`1`

4

`%score`

5

`num_people`

6

`num_people_who_have_visited_boston_in_2016`

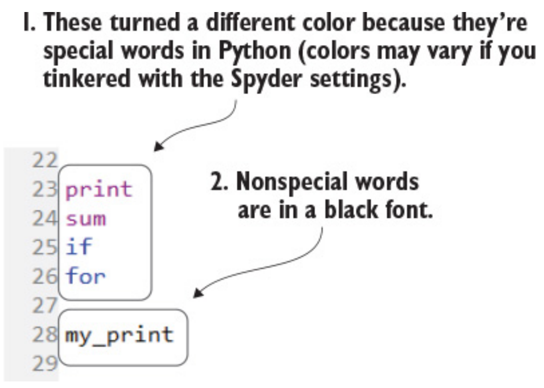
Programming languages have a few reserved words that you can't use as variable names. For Python, Spyder has *syntax highlighting*, which changes the color of words that are special reserved Python *keywords*.

Definition

A keyword is a special word. It's reserved because it has special meaning in a programming language.

Figure 4.2 shows an example of syntax highlighting. A good general rule is that if the variable you want to use turns a different color, you shouldn't use it as a variable name.

Figure 4.2. Special words that have a meaning in Python change color in the code editor. As a general rule, you shouldn't name your variables using any words that turn a color other than black.


1. These turned a different color because they're special words in Python (colors may vary if you tinkered with the Spyder settings).
2. Nonspecial words are in a black font.

```
print
sum
if
for

my_print
```

In addition to the preceding rules for naming variables, here are some guidelines to help you write programs that are more readable:

- Choose descriptive and meaningful names instead of short, single-character names.
- Use underscores to add a pretend space between variable words.
- Don't use variable names that are too long.
- Be consistent throughout your code.

Quick check 4.4

Are the following allowed and good variable names?

1

`customer_list`

2

`print` (where this word is a color other than black in Spyder)

3

`rainbow_sparkly_unicorn`

4

`list_of_things_I_need_to_pick_up_from_the_store`

4.2.4. Creating a variable

Before you can work with a variable, you have to set it to a value. You *initialize* the variable by assigning it to an object, using the equal sign. The initialization binds the object to the variable name.

Definition

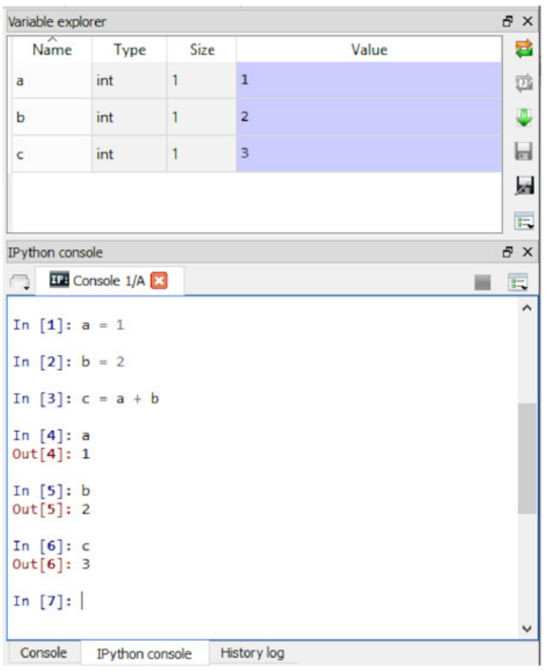
A variable initialization binds a variable name to an object.

After you initialize a variable, you can refer to a particular object by using its variable name. In Spyder, type the following lines in the console to initialize three variables:

```
a = 1
b = 2
c = a + b
```

You can use the variable explorer to see the names of variables, their type, and size (you'll see what this means in following lessons), and their value. Figure 4.3 shows how your screen should look.

Figure 4.3. How to create variables in the console. The variable explorer shows you what variables you have set up and initialized in this session.



Variable explorer

Name	Type	Size	Value
a	int	1	1
b	int	1	2
c	int	1	3

IPython console

Console 1/A

```
In [1]: a = 1

In [2]: b = 2

In [3]: c = a + b

In [4]: a
Out[4]: 1

In [5]: b
Out[5]: 2

In [6]: c
Out[6]: 3

In [7]: |
```

You should see that the variable explorer is populated with the variables you create and their values. If you type in the name of a variable in the console and hit Enter, this allows you to *peek* into its value. The variable explorer also tells you an additional bit of information in the second column: the type of the variable. The next section goes into more detail on what this means.

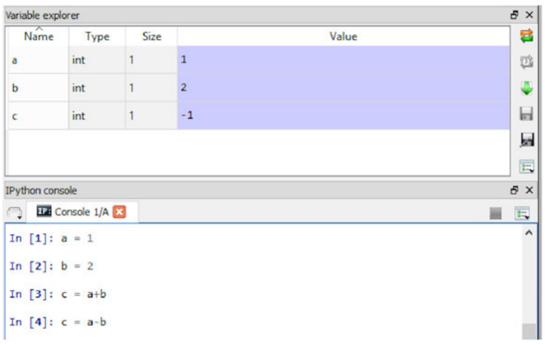
4.2.5. Updating a variable

After you create a variable name, you can update the name to be any object. You saw that these lines initialize three variables:

```
a = 1
b = 2
c = a + b
```

You can update the value of `c` to be something else. Now you can type `c = a - b` to reassign the variable `c` to have a new value. In the variable explorer, you should see that the variable `c` now has a different value. Figure 4.4 shows how Spyder looks now.

Figure 4.4. The variable explorer has the same variable c, except with a new value.



Name	Type	Size	Value
a	int	1	1
b	int	1	2
c	int	1	-1

```
In [1]: a = 1

In [2]: b = 2

In [3]: c = a+b

In [4]: c = a-b
```

Variable names merely bind names to objects. The same name can be reassigned to a different object. A Python operation, named `id`, shows the identity of an object in the form of a sequence of digits. The identity is unique for every object and won't change while the object exists. Type the following lines in the console:

```
c = 1
id(c)
c = 2
id(c)
```

After the first `id(c)` command, my console printed out `1426714384`. After the second `id(c)` command, I got `1426714416`. These are two numbers for the same variable name because the numbers 1 and 2 are different objects.

Quick check 4.5

Assume you're doing the following actions in order. Write a line of code for each:

1

Initialize a variable named `apples` to the value `5`.

2

Initialize a variable named `oranges` to the value `10`.

3

Initialize a variable named `fruits` to the sum of `apples` and `oranges`.

4

Reassign the variable `apples` to be `20`.

5

Recalculate the variable `fruits` just as before.

SUMMARY

In this lesson, my objective was to teach you

- To create and initialize variables
- That not all names are allowed for variable names and that there are general rules for naming your variables
- That an object has a value
- That expressions are lines of code that can be reduced to a value
- That an object has operations you can do on it
- That a variable is a name that is bound to an object

Let's see if you got this ...

Q4.1

```
a = 2
b = 2
a + x = b
```

Q4.2

Type `a + x = b` in the Spyder console and hit Enter. You get an error. Maybe the error happened because you didn't tell the computer what `a` and `b` were. Type the following lines in the console, each followed by pressing Enter. Do you still get an error?

```
a = 2
b = 2
a + x = b
```

Recommended / Playlists / History / Topics / Settings / Get the App / Sign Out

PREV
Lesson 3. Introducing Python: a programming language

NEXT
Lesson 5. Object types and statements of code