Lesson 5. Object types and statements of code

After reading lesson 5, you'll be able to

- Write code that creates various types of objects
- Write simple lines of code to manipulate Python variables

Suppose you have a family as follows:

- *Four people*—Alice, Bob, Charlotte, and David
- *Three cats*—Priss, Mint, and Jinx
- *Two dogs*—Rover and Zap

Every person, cat, and dog is a separate object. You named each object something different so you can easily refer to them and so that everyone else knows which object you're talking about. In this family, you have three types of objects: people, cats, and dogs.

Each type of object has characteristics that are different from another type. People have hands and feet, whereas cats and dogs have only feet. Cats and dogs have whiskers, but people don't. The characteristics of a type of object uniquely identify all individual objects in that type. In programming, characteristics are called data *attributes*, or *values*, for the type.

Each type of object also has actions or behavior. People can drive a car, but dogs and cats can't. Cats can climb trees, but dogs can't. The actions that a type of object can do are specific to that object only. In programming, actions are called *operations* on the type.

Consider this

You have a sphere and a cube. Write some characteristics of each (choose characteristics that uniquely identify them) and some actions you can do with each.

Answer:

Sphere—Round, has a radius/diameter, rolls, bounces

Cube—All sides equal length, stays flat, has points, can stand on it

5.1. TYPES OF THINGS

So far, you've created variables to store objects. Variables are names given to individual objects. In reality, you can classify objects into groups. All objects in the same group are going to be of the same type; they'll all have the same basic properties, and they'll all have the same basic operations for interacting with them.

5.2. BASIC TYPE OF OBJECTS IN PROGRAMMING

Objects have

- A type, which dictates the values they can have
- Operations you can do with them

Definition

An object type tells you what kinds of values the object can have.

In most programming languages, a few types of objects are the basic building blocks for each language. These types of objects might be called *primitives*, or *scalars*. These basic types are built in to the language, and every

of the English language; from the 26 letters, you can make words, sentences, paragraphs, and novels.

Python has five basic *types* of objects: integers, floating point, Booleans, strings, and a special type that represents the absence of a value. In Python, these five types are called *primitives*, and every other type of object in the language can be constructed with these five types.

5.2.1. Integers as whole numbers

An object of type *integer* (in Python, the `int` type) is an object whose values are real whole numbers. For example, 0, 1, 2, 5, 1234, -4, -1000 are all integers.

The kinds of operations you can do on these numbers are, as you might expect, operations that you would do on numbers in math class.

You can add, subtract, multiply, and divide two or more numbers. You may have to surround a negative number with parentheses to avoid confusing the negative number with the subtraction operation. For example,

- `a = 1 + 2` adds an integer object with value `1` and an integer object with value `2` together and binds the resulting object with a value of `3` to the variable named `a`.
- `b = a + 2` adds the value of the integer object named `a` and the integer object with value `2` together and binds the resulting object's value to the variable named `b`.

You can increment a number by a certain value. For example,

- `x = x + 1` means that you add `1` to the value of `x` and rebind that value to the variable named `x`. Notice that this is different than in math, where you would solve this equation by moving `x` from the right to the left of the equal sign (or subtracting `x` from both sides of the equation) and simplify the expression to `0 = 1`.
- `x += 1` is programming shorthand notation for `x = x + 1`. It's alternate, valid syntax. You can replace the `+=` with `*=`, `-=`, or `/=` to stand for `x = x * 1`, `x = x - 1`, or `x = x / 1`, respectively. The `1` on the right-hand side of the equal sign can also be replaced with any other value.

Quick check 5.1

Write a line of code that achieves each of the following:

1

Add `2` and `2` and `2` and store the result in a variable named `six`.

2

Multiply the variable `six` with `-6` and store the result in a variable named `neg`.

3

Use shorthand notation to divide the variable `neg` by `10` and store the result in the same variable, `neg`.

5.2.2. Floating point as decimal numbers

An object of type *floating point* (in Python, the `float` type) is an object whose values are decimal numbers. For example, 0.0, 2.0, 3.1415927, and -22.7 are all floats. If you've played around with integers, you may have noticed that when you divided two numbers, the result was a float type. The kinds of operations you can do on these numbers are the same as those for integers.

It's important to understand that the following two lines lead to two variables representing objects of two types. The variable `a` is an integer, but the variable `b` is a float:

```
a = 1
b = 1.0
```

Quick check 5.2

Write a line of code that achieves each of the following:

1

2

Subtract the variable `half` from `1.0` and store the result in a variable named `other_half`.

5.2.3. Booleans as true/false data

In programming, you often work with more than just numbers. A type of object that's even simpler than a number is the *Boolean* (in Python, the `bool` type); it has only two possible values, `True` or `False`. They replace expressions with one of these two values; for example, the expression `4 < 5` is replaced by `False`. The kinds of operations you can do on Booleans involve the logic operations `and` and `or` and were briefly introduced in lesson 1.

Quick check 5.3

Write a line of code that achieves each of the following:

1

Store the value `True` in a variable named `cold`.

2

Store the value `False` in a variable named `rain`.

3

Store the result of the expression `cold and rain` in a variable named `day`.

5.2.4. Strings as sequences of characters

A useful data type is the string (in Python, the `str` type), which is covered in a lot more detail in lessons 7 and 8. Briefly, a *string* is a sequence of characters surrounded by quotation marks.

One character is anything you can enter by hitting one key on the keyboard. The quotations surrounding characters can either be single or double quotation marks (`'` or `"`) as long as you're consistent. For example, `'hello'`, `"we're # 1!"`, `"m.ss.ng c.ns.n.nts??"`, and `"'"` (the single quote inside two double quotes) are possible values for a string.

You can do many operations on strings, and they are detailed in lesson 7.

Quick check 5.4

Write a line of code that achieves each of the following:

1

Create a variable named `one` with the value `"one"`.

2

Create a variable named `another_one` with the value `"1.0"`.

3

Create a variable named `last_one` with the value `"one 1"`.

5.2.5. The absence of a value

You might want to designate the absence of a value in a program. For example, if you get a new pet and haven't named it yet, the pet doesn't have a value for its name. Programming languages allow you to designate a special value in this situation. In many programming languages, this is referred to as `null`. In Python, the value is `None`. Because in Python everything is an object, this `None` also has a type, `NoneType`.

Quick check 5.5

What is the type of each object?

1

2

```
27
```

3

```
False
```

4

```
"False"
```

5

```
"0.0"
```

6

```
-21
```

7

```
99999999
```

8

```
"None"
```

9

```
None
```

5.3. WORKING WITH BASIC TYPES OF DATA VALUES

Now that you understand a little bit about the types of objects you'll be working with, you can start to write code consisting of more than one line of code. When you're writing a program, each line of code is called a *statement*. A statement may or may not contain an *expression*.

Definition

A statement is any line of code.

5.3.1. Building blocks of expressions

An *expression* is an operation between objects that can be reduced to a single value. The following are examples of expressions (and statements):

- `3 + 2`
- `b - c` (if you know the values of `b` and `c`)
- `1 / x`

A line of code that prints something is a statement but not an expression, because the act of printing can't be reduced to a value. Similarly, a variable assignment is an example of a Python statement but not an expression, because the act of doing the assignment doesn't have a value.

Quick check 5.6

Write down whether each of the following is a statement, expression, or both:

1

```
2.7 - 1
```

2

```
0 * 5
```

3

```
a = 5 + 6
```

4

5.3.2. Converting between different types

If you're not sure of the type of an object, you can use Spyder to check for yourself. In the console, you can use a special command, `type()`, to get the data type of an object. For example,

- Type in the console `type(3)` and hit Enter to see that the type of `3` is an integer.
- Type in the console `type("wicked")` and hit Enter to see that the type of `"wicked"` is a string.

You can also convert objects from one type to another. To do this in Python, you surround the object you want to convert with parentheses and the name of the type that you want to convert to. For example,

- `float(4)` gives `4.0` by converting the int `4` to the float `4.0`.
- `int("4")` gives `4` by converting the string `"4"` to the int `4`. Notice that you can't convert a string that isn't a number to an int or a float. If you try to convert `int("a")`, you'll get an error.
- `str(3.5)` gives `"3.5"` by converting the float `3.5` to the string `"3.5"`.
- `int(3.94)` gives `3` by converting the float `3.94` to the int `3`. Notice this truncates the number to keep only the whole number before the decimal point.
- `int(False)` gives `0` by converting the bool `False` to the int `0`. Notice that `int(True)` gives `1`.

Quick check 5.7

Write an expression that converts the following objects to the desired types and then predict the converted value. Remember that you can check by typing the expressions in the Python console:

1

`True` to a `str`

2

`3` to a `float`

3

`3.8` to a `str`

4

`0.5` to an `int`

5

`"4"` to an `int`

5.3.3. How arithmetic impacts object types

Mathematical operations are one example of Python expressions. When you do an operation between numbers in Python, you get a value, so all mathematical operations are expressions in Python.

Many of the operators between numbers in math work between numbers (ints or floats) in Python. You're allowed to mix and match integers and floats when you do the mathematical operations. Table 5.1 shows what happens when you do mathematical operations on all possible combinations of ints and floats. The first row of the table says that when you add an int to another int, you get an int result. One example of this is adding 3 and 2 to get 5. When at least one of the operands is a float, the result will be a float. Adding 3.0 + 2, or 3 + 2.0, or 3.0 + 2.0 all result in the float 5.0. The exception to this is division. When you divide two numbers, you always get a float, no matter what the operand types are.

Table 5.1 shows two operations you haven't seen before—the power and the remainder:

- The *power* (`**`) is a base raised to the power of an exponent. For example, 3^2 is written as `3 ** 2` in Python.
- The *remainder* (`%`) gives you the remainder when the first object is divided by the second object. For example, `3 % 2` finds out how many times 2 goes into 3 (only one time) and then tells you how much is left

Table 5.1. Mathematical operations on ints and floats and the resulting types

Type of first object	Operation(s)	Type of second object	Type of result	Example	
int	+ - * ** %	int	int	3 + 2 3 - 2 3 * 2 3 ** 2 3 % 2	gives 5 gives 1 gives 6 gives 9 gives 1
int	/	int	float	3 / 2	gives 1.5
int	+ - * / ** %	float	float	3 + 2.0 3 - 2.0 3 * 2.0 3 / 2.0 3 ** 2.0 3 % 2.0	gives 5.0 gives 1.0 gives 6.0 gives 1.5 gives 9.0 gives 1.0
float	+ - * / ** %	int	float	3.0 + 2 3.0 - 2 3.0 * 2 3.0 / 2 3.0 ** 2 3.0 % 2	gives 5.0 gives 1.0 gives 6.0 gives 1.5 gives 9.0 gives 1.0
float	+ - * / ** %	float	float	3.0 + 2.0 3.0 - 2.0 3.0 * 2.0 3.0 / 2.0 3.0 ** 2.0 3.0 % 2.0	gives 5.0 gives 1.0 gives 6.0 gives 1.5 gives 9.0 gives 1.0

Another operation you can do on numbers is to round them by using the `round()` command; for example, `round(3.1)` gives you the int `3`, and `round(3.6)` gives you the int `4`.

Quick check 5.8

What is the value and type of the resulting value of each expression? Recall that you can use the `type()` command to check yourself. You can even put an expression inside the parentheses of `type`; for example, `type(3 + 2)`.

1

```
2.25 - 1
```

2

```
3.0 * 3
```

3

```
2 * 4
```

4

```
round(2.01 * 100)
```

5

```
2.0 ** 4
```

6

```
2 / 2.0
```

7

```
6 / 4
```

8

```
6 % 4
```

9

```
4 % 2
```

SUMMARY

In this lesson, my objective was to teach you about variables with a few basic types in Python: integers, floats, Booleans, strings, and a special `NoneType`. You wrote code to work with object types, and to do specific operations on ints and floats. You also wrote statements and expressions. Here are the ma-

- All expressions are statements, but not all statements are expressions.
- Basic data types are ints, floats, bools, strings, and a special type to represent the absence of a value.

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

PREV
Lesson 4. Variables and expressions: giving names and value…

NEXT
Lesson 6. Capstone project: your first Python program—con…