PREV
Unit 7. Making your own object types by using obj

NEXT
Lesson 31. Creating a class for an object type

Lesson 30. Making your own object types

After reading lesson 30, you'll be able to

- Understand that an object has properties
- Understand that an object has operations associated with it
- Understand what dot notation means when working with objects

You use objects all the time in your daily life. You use computers and phones, handle boxes and envelopes, and interact with people and animals. Even numbers and words are basic objects.

Every object you use is made up of other objects. Except for the basic building blocks of matter, every object you interact with can be decomposed into smaller objects. For example, your calculator can be decomposed into a few basic components: the logic chip, screen, and buttons (and each of these into smaller components). Even a sentence can be decomposed into individual words arranged in a certain order.

Every object you interact with has certain behaviors. For example, a basic calculator can do mathematical operations but can't check email. The calculator has been programmed to work in a certain way depending on which key or button is pressed. Words in different languages can be arranged differently, according to the rules of the language, to form sentences that make sense.

As you build complex systems, you can reuse objects you've already built without going back to the basic building blocks of matter. For example, a computer may have the same logic chip that a calculator already has, to do basic arithmetic. In addition to that, a computer may also have components already built into it that allow it to access the internet or to display color graphics.

The same idea can be applied to programming! You can create more-complex object types to use in your programs, made up from other object types. In fact, you may have noticed that lists and dictionaries are object types that are made up of other object types: a list contains a set of objects, and a dictionary contains a set of pairs of objects.

Consider this

Here are some properties and behaviors of two objects. Can you separate properties from behaviors? What are the objects?

- Two eyes
- Sleeps on a keyboard
- No eyes
- Any color
- Scratches
- Bounces
- Fur
- Round
- Rolls
- Hides
- Four limbs

Answer:

- A cat.
- Characteristics: Two eyes, fur, four limbs
- Behaviors: Sleeps on a keyboard, scratches, hides

- Behaviors: Bounces, rolls

30.1. WHY DO YOU NEED NEW OBJECT TYPES?

You've been working with object types since you wrote your first line of code. Integers, floats, strings, Booleans, tuples, lists, and dictionaries are all types of objects. They're objects that are built into the Python language, meaning that they're available to use by default when you start Python. As you were working with lists (and dictionaries), you may have noticed that they're object types made up of other object types. For example, the list `L = [1,2,3]` is a list made up of integers.

Integers, floats, and Booleans are atomic objects because they can't be separated into smaller object types; these types are the basic building blocks of the Python language. Strings, tuples, lists, and dictionaries are nonatomic objects because they can be decomposed into other objects.

Using different object types helps organize your code and make it more readable. Imagine how confusing code would look if all you had to use were the atomic data types. If you wanted to write code that contained your grocery list, you might have to create a string variable for each of the list items. That would quickly make your program messy. You'd have to make variables as you realize you have more items to add.

As you continue to build more complex programs, you'll find that you want to create your own object types. These object types "save" a set of properties and a set of behaviors under this new type of object. The properties and behaviors are things that you, as the programmer, get to decide on and define. As you build programs, you can create new object types from other types, even ones that you create yourself.

Quick check 30.1

For each of the following scenarios, would you need to create a new object type or can you represent it with an object type you already know?

1

Someone's age

2

Latitude and longitude of a group of map points

3

A person

4

A chair

30.2. WHAT MAKES UP AN OBJECT?

An object type is defined by two things: a set of properties and a set of behaviors.

30.2.1. Object properties

Object type *properties* are data that define your object. What characteristics can be used to explain the "look" of your object?

Let's say you want to create an object type that represents a car. What data can describe a generic car? As the creator of the car type, you get to decide on how much or how little data defines the generic car. The data can be things like the length, width, height, or the number of doors.

After you decide on the properties for a specific object type, these choices will define your type and will be fixed. When you start adding behaviors to your type, you may manipulate these properties.

Here are a few more examples of properties for object types. If you have a circle type, its data may be its radius. If you have a "point on map" type, the data may be the values of the latitude and longitude. If you have a room type, its data may be its length, width, height, number of items that are in it, and whether it has an occupant.

Quick check 30.2

What are some appropriate data you may use to represent each of the following types?

Rectangle

2

TV

3

Chair

4

Person

30.2.2. Object behaviors

Object type *behaviors* are operations that define your object. What are some ways that someone can interact with your type?

Let's go back to the generic car type. How can someone interact with a car? Once again, as the creator of the car object, you get to decide the number of ways you'll allow someone to interact with it. A car's behaviors may be things like changing the color of the car, getting the car to make a noise, or making the car's wheels turn.

These operations are actions that objects of this type, and only this type, can do. These can be actions done by the objects themselves, or ways that an object can interact with other objects.

How do other object types behave? For a circle, one action could be to get its area or its circumference. For a point on a map, one action could be to get the country it's in and another action could be to get the distance between two points. For a room, one action might be to add an item, which increases the item count by 1, or to remove an item to decrease the item count, and another could be to get the volume of the room.

Quick check 30.3

What are some appropriate behaviors you may add for each of the following object types?

1

Rectangle

2

TV

3

Chair

4

Person

30.3. USING DOT NOTATION

You already have an idea of what an object type is. An object type has properties and operations. Here are some object types that you've already worked with:

- An integer is a whole number. Its operations are addition, subtraction, multiplication, division, casting to a float, and many more.
- A string is a sequence of characters. Its operations are addition, indexing, slicing, finding a substring, replacing a substring by another, and many more.
- A dictionary has a key, a value, and a formula to map a key to a memory location to put the value there. Its operations are getting all the keys, getting all the values, indexing using a key, and many more.

Properties and behaviors are defined for, and belong to, a particular object type; other object types don't know about them.

In lesson 7, you used dot notation on strings. Dot notation indicates that you're accessing data or behaviors for a particular object type. When you use dot notation, you indicate to Python that you want to either run a particular operation on, or to access a particular property of, an object type. Python list named `L`, you appended an item to the list with `L.append()`. The dot notation leads Python to look at the object, `L`, that the operation, `append`, is being applied to. Python knows that `L` is of type `list` and checks to make sure that the `list` object type has an operation named `append` defined.

Quick check 30.4

In the following examples of dot notation, on what object type is the operation being done?

1

```
"wow".replace("o", "a")
```

2

```
[1,2,3].append(4)
```

3

```
{1:1, 2:2}.keys()
```

4

```
len("lalala")
```

SUMMARY

In this lesson, my goal was to teach you that an object type is represented by two things: its data properties and its behaviors. You've been using objects built into Python and have even seen dot notation used on more-complex types including strings, lists, and dictionaries. Here are the major takeaways:

- An object type has data properties: other objects that make up the type.
- An object type has behaviors: operations that allow interactions with objects of this type.
- Objects of the same type know the properties and behaviors that define them.
- Dot notation is used to access properties and behaviors of an object.

PREV
Unit 7. Making your own object types by using object-orient…

NEXT
Lesson 31. Creating a class for an object type