Lesson 24. Mutable and immutable objects

After reading lesson 24, you'll be able to

- Understand what an immutable object is
- Understand what a mutable object is
- Understand how objects are stored in computer memory

Consider the following scenario. You buy a house that's just the right size for you; it's big enough for one person to live in. Then you get married, and there's no space for your spouse. You have two choices: build an addition to the house, or tear the whole house down and build a new larger house to hold two people. Building an addition makes more sense than demolishing a perfectly good house and making an exact copy of it just to make an addition. Now, you have a kid and decide you need more room. Again, do you build an addition or tear the house down to build a new one to hold three people? Again, it makes more sense to build an extension. As you're adding more people to your house, it's quicker and less costly to keep the same structure and modify it.

In some situations, it helps to be able to put your data in some sort of container so you can modify the data within the container instead of having to create a new container and put the modified data in the new one.

Consider this

This exercise requires a piece of paper and your computer. Think of the names of all the countries you've visited. If you need inspiration, suppose you've visited Canada, Brazil, Russia, and Iceland:

- On a piece of paper, use a pen to write all the countries you've visited in alphabetical order, one on each line.
- On a text editor on your computer, type the same list of countries.
- Suppose you realize that you visited Greenland, not Iceland. Modify your list on paper so you have Canada, Brazil, Russia, and Greenland in alphabetical order. Can you modify the list (keeping one country per line) without having to rewrite it all? Can you modify the list on the computer (keeping one country per line) without having to rewrite it all?

Answer:

Using a pen, I'd have to rewrite the list. Otherwise, it'd be too messy to scratch out and write the new country name beside it. On the text editor, I can replace the name directly.

24.1. IMMUTABLE OBJECTS

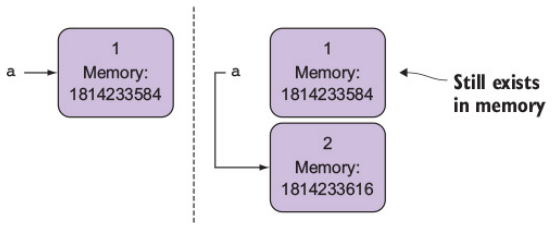
All Python objects that you've seen (Booleans, integers, floats, strings, and tuples) are immutable. After you create the object and assign a value to it, you can't modify that value.

Definition

An *immutable object* is an object whose value can't change.

What does this mean behind the scenes, in computer memory? An object created and given a value is assigned a space in memory. The variable name bound to the object points to that place in memory. Figure 24.1 shows the memory locations of objects and what happens when you bind the same variable to a new object by using the expressions `a = 1` and then `a = 2`.

Figure 24.1. The variable named a is bound to an object with value 1 in one memory location. When the variable named a is bound to a different object with value 2, the original object with value 1 is still in memory; you just can't access it anymore through a variable.



You can see the value of the memory location to which the object has been assigned, using the `id()` function. Type the following in the console:

```
a = 1
id(a)
```

The value shown represents the location in memory of the object with value `1`, accessed by the variable named `a`. Now, type the following:

```
a = 2
id(a)
```

As before, the value shown represents the location in memory of the object with value `2`, accessed by the variable named `a`. Why are these values different if you're using variable name `a` in both cases? We come back to the idea that the variable name is a name bound to an object. The name points to an object; in the first case, the variable points to the integer object with value `1` and then to the object with value `2`. The `id()` function tells you the memory location of the object pointed to by the variable name, not anything about the variable name itself.

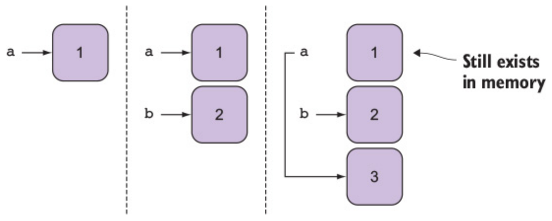
Objects of the types you've seen so far can't be modified after they're created. Suppose you have the following lines of code, which are executed in the order shown. You initialize two variables, `a` and `b`, to two objects with values `1` and `2`, respectively. Then you change the binding of variable `a` to a different object with a value of `3`:

```
a = 1
b = 2
a = 3
```

Figure 24.2 shows the objects that exist in your program's memory with each line of code:

- When you create the object with value `1`, you bind the object to the variable named `a`.
- When you create the object with value `2`, you bind the object to the variable named `b`.
- In the final line, you're rebinding the variable name `a` to a completely new object, one whose value is `3`.

Figure 24.2. Progression of binding variables to objects. On the left, `a = 1` shows that object 1 is at some memory location. In the middle, `a = 1` and then `b = 2`. Objects with values 1 and 2 are at different memory locations. On the right, `a = 1` and then `b = 2` and then `a = 3`. The variable named a is bound to a different object, but the original object still exists in memory.



The old object with a value of `1` may still exist in computer memory, but you lost the binding to it; you don't have a variable name as a way to refer to it anymore.

After an immutable object loses its variable handle, the Python interpreter may delete the object to reclaim the computer memory it took up and use it for something else. Unlike some other programming languages, you (as the programmer) don't have to worry about deleting old objects; Python takes care of this for you through a process called *garbage collection*.

Q1:

Draw a diagram similar to the one in figure 24.2 to show variables and objects that they point to (and any leftover objects) for the following sequence of statements:

```
sq = 2 * 2
ci = 3.14
ci = 22 / 7
ci = 3
```

24.2. THE NEED FOR MUTABILITY

After you lose the variable binding to an object, there's no way to get back to that object. If you want the program to remember its value, you need to store its value in a temporary variable. Using a temporary variable to store values that you don't need right now, but may need in the future, isn't an efficient way of programming. It wastes memory and leads to cluttered code filled with variables that, for the most part, will never be used again.

If immutable objects are objects whose value can't change after they're created, a mutable object is an object whose value can change after it's created. Mutable objects are often objects that can store a collection of data. In later lessons in this unit, you'll see lists (Python type `list`) and dictionaries (Python type `dict`) as examples of mutable objects.

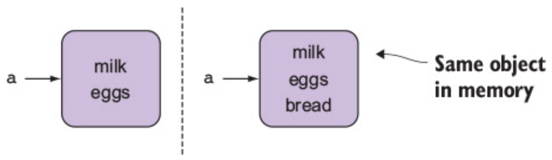
Definition

A *mutable object* is an object whose value can change.

For example, you can make a list of items you'll need from the grocery store; as you decide what you need, you add items to the list. As you buy things, you remove them from the list. Notice that you're using the same list and modifying it (crossing out or adding to the end), as opposed to having many lists in which you copy over items every time you want to make a change. As another example, you can keep your grocery needs in a dictionary that maps every item you need from the store to a number representing the quantity that you need.

Figure 24.3 shows what happens in memory when you bind variables to mutable objects. When you modify the object, you keep the same variable binding, and the object at the same memory location is directly modified.

Figure 24.3. On the left, you have a grocery list at a certain memory location. On the right, you add another item to your grocery list, and the object at the same memory location is directly modified.



Mutable objects are more flexible when programming, because you can modify the object itself without losing the binding to it.

First, a mutable object can behave the same way as an immutable object. If you rebind a sample grocery list to a variable a and check its memory location, you see that the memory location changes and the binding to the original list is lost:

```
a = ["milk", "eggs"]
id(a)
a = ["milk", "eggs", "bread"]
id(a)
```

But you have the option to modify the original object directly without losing the binding to it, using operations that work only on mutable objects. In the following code, you append one more item (add it to the end of the list). The memory location of the object that the variable a is bound to remains unchanged. The behavior of the following code is shown in figure 24.3:

```
a = ["milk", "eggs"]
id(a)
a.append("bread")
id(a)
```

First, you can store data that's part of a collection (for example, lists of people or mappings of people to phone numbers) in an object, and you can keep the object for use later.

After the object is created, you can add data to and remove data from the object itself, without creating a new object. When you have the object, you can also modify elements in the collection by modifying the elements in the object itself instead of creating a new copy of the object with only one of its values modified.

Finally, you can rearrange data in the collection by keeping the same object and making the rearrangement in place—for example, if you have a list of people and you want to sort it alphabetically.

With large collections of data, copying your collection into a new object every time you make a change to it would be inefficient.

Quick check 24.2

Would you use a mutable or an immutable type of object to store the following information?

1

Cities in a state

2

Your age

3

Group of items in a grocery store and their cost

4

Color of a car

SUMMARY

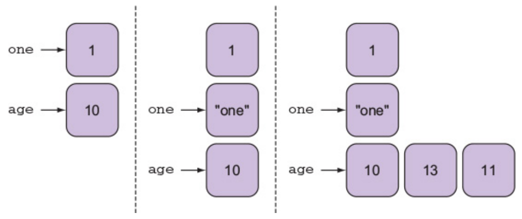
In this lesson, my objective was to teach you how an object exists in computer memory. The values of some objects can't change after they're created (immutable objects). The values of some objects can change after they're created (mutable objects). You may use one kind or another kind of object, depending on the task you're trying to accomplish using programming. Here are the major takeaways:

- Immutable objects can't change their value (for example, strings, integers, floats, Booleans).
- Mutable objects can change their value (in this unit, you'll see lists and dictionaries).

Let's see if you got this...

Q24.1

In the following diagram, each panel is showing a new operation of code. Which of the following variables are bound to immutable objects? Which are bound to mutable objects?



Progression of expressions being added to code that manipulates two variables: `one` and `age`