15.1. Introduction: Optional Parameters

In the treatment of functions so far, each function definition specifies zero or more formal parameters and each function invocation provides exactly that many values. Sometimes it is convenient to have **optional parameters** that can be specified or omitted. When an optional parameter is omitted from a function invocation, the formal parameter is bound to a **default value**. When the optional parameter is included, then the formal parameter is bound to the value provided. Optional parameters are convenient when a function is almost always used in a simple way, but it's nice to allow it to be used in a more complex way, with non-default values specified for the optional parameters.

Consider, for example, the `int` function, which you have used previously. Its first parameter, which is required, specifies the object that you wish to convert to an integer. For example, if you call in on a string, `int("100")`, the return value will be the integer 100.

That's the most common way programmers want to convert strings to integers. Sometimes, however, they are working with numbers in some other "base" rather than base 10. For example, in base 8, the rightmost digit is ones, the next digit to the left is 8s, and the one to the left of that is the 64s place (8**2).

The int function provides an optional parameter for the base. When it is not specified, the number is converted to an integer assuming the original number was in base 10. We say that 10 is the default value. So `int("100")` is the same as invoking `int("100", 10)`. We can override the default of 10 by supplying a different value.

Save & Run | Original - 1 of 1 | Show in CodeLens

```
1 print(int("100"))
2 print(int("100", 10))    # same thing, 10 is the default value for the base
3 print(int("100", 8))     # now the base is 8, so the result is 1*64 = 64
4
```

```
100
100
64
```

Activity: 1 -- ActiveCode (ac15_1_1)

Note

Tom Lehrer's New Math

Some math educators believe that elementary school students will get a much deeper understanding of the place-value system, and set a foundation for learning algebra later, if they learn to do arithmetic not only in base-10 but also in base-8 and other bases. This was part of a movement called "The New Math", though it's not so new now (I had it when I was in elementary school!) Tom Lehrer made a really funny song about it, and it's set with visuals in several YouTube renditions now. Try this very nice lip-synched version.

When defining a function, you can specify a default value for a parameter. That parameter then becomes an optional parameter when the function is called. The way to specify a default value is with an assignment statement inside the parameter list. Consider the following code, for example.

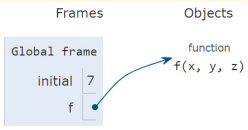
Python 3.3

```
1 initial = 7
2 def f(x, y =3, z=initial):
3     print("x, y, z, are: " + str(x) + ", " + str(y) + ", " + str(z))
4
5 f(2)
6 f(2, 5)
7 f(2, 5, 8)
```

<< First | < Back | Program terminated | Forward > | Last >>

line that has just executed
next line to execute

Visualized using Online Python Tutor by Philip Guo



Program output:

```
x, y, z, are: 2, 3, 7
x, y, z, are: 2, 5, 7
x, y, z, are: 2, 5, 8
```

Activity: 2 -- CodeLens: (clens15_1_1)

Notice the different bindings of x, y, and z on the three invocations of f. The first time, y and z have their default values, 3 and 7. The second time, y gets the value 5 that is passed in, but z still gets the default value of 7. The last time, z gets the value 8 that is passed in.

If you want to provide a non-default value for the third parameter (z), you also need to provide a value for the second item (y). We will see in the next section a mechanism called keyword parameters that lets you specify a value for z without specifying a value for y.

Note

This is a second, related but slightly different use of = than we have seen previously. In a stand-alone assignment statement, not part of a function definition, `x=3` assigns 3 to the variable x. As part of specifying the parameters in a function definition, `x=3` says that 3 is the *default* value for x, used *only when* no value is provided during the function invocation.

There are two tricky things that can confuse you with default values. The first is that the default value is determined at the time that the function is defined, not at the time that it is invoked. So in the example above, if we wanted to invoke the function f with a value of 10 for z, we cannot simply set initial = 10 right before invoking f. See what happens in the code below, where z still gets the value 7 when f is invoked without specifying a value for z.

Python 3.3

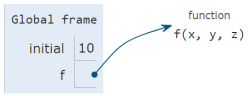
```
1 initial = 7
2 def f(x, y =3, z=initial):
3     print("x, y, z, are: " + str(x) + ", " + str(y) + ", " + str(z))
4
5 initial = 10
6 f(2)
```

<< First | < Back | Program terminated | Forward > | Last >>

line that has just executed
next line to execute

Visualized using Online Python Tutor by Philip Guo

Frames Objects



Program output:

```
x, y, z, are: 2, 3, 7
```

Activity: 3 -- CodeLens: (clens15_1_2)

The second tricky thing is that if the default value is set to a mutable object, such as a list or a dictionary, that object will be shared in all invocations of the function. This can get very confusing, so I suggest that you never set a default value that is a mutable object. For example, follow the exceution of this one carefully.

Python 3.3

```
1 def f(a, L=[]):
2     L.append(a)
3     return L
4
5 print(f(1))
6 print(f(2))
7 print(f(3))
8 print(f(4, ["Hello"]))
9 print(f(5, ["Hello"]))
```

<< First | < Back | Program terminated | Forward > | Last >>

line that has just executed
next line to execute

Visualized using Online Python Tutor by Philip Guo

Frames Objects

Program output:

```
[1]
[1, 2]
[1, 2, 3]
['Hello', 4]
['Hello', 5]
```

Activity: 4 -- CodeLens: (opt_params_4)

When the default value is used, the same list is shared. But on lines 8 and 9 two different copies of the list [“Hello”] are provided, so the 4 that is appended is not present in the list that is printed on line 9.

Check your understanding

advfuncs-1-1: What will the following code print?

```
def f(x = 0, y = 1):
    return x * y

print(f())
```

- A. 0
- B. 1
- C. None
- D. Runtime error since no parameters are passed in the call to f.

Check me | Compare me

✔ Since no parameters are specified, x is 0 and y is 1, so 0 is returned.

Activity: 5 -- Multiple Choice (question15_1_1)

advfuncs-1-2: What will the following code print?

```
def f(x = 0, y = 1):
    return x * y

print(f(1))
```

- A. 0
- B. 1
- C. None
- D. Runtime error since the second parameter value is missing.

Check me | Compare me

✔ Since one parameter value is specified, it is bound to x; y gets the default value of 1.

Activity: 6 -- Multiple Choice (question15_1_2)

3. Write a function called `str_mult` that takes in a required string parameter and an optional integer parameter. The default value for the integer parameter should be 3. The function should return the string multiplied by the integer parameter.

Save & Run | 5/14/2021, 3:59:39 PM - 4 of 4 | Show in CodeLens

```
1 def str_mult(a,b=3):
2     return a*b
3
```

Activity: 7 -- ActiveCode (ac15_1_2)

Result	Actual Value	Expected Value	Notes
Pass	'hahaha'	'hahaha'	Testing that str_mult('ha') returns 'hahaha'
Pass	'hahah...ahaha'	'hahah...ahaha'	Testing that str_mult('ha') returns 'hahahahahahahahahaha'
Pass	''	''	Testing that str_mult('ha', 0) returns ''

Expand Differences

You passed: 100.0% of the tests

You have attempted 8 of 7 activities on this page

15. Advanced Functions">
15.2. Keyword Parameters">
anced Functions">
✔ Completed. Well Done!
15.2. Keyword Parameters">Next Section - 15.2. Keyword Parameters

| Back to top