15.2. Keyword Parameters

In the previous section, on Optional Parameters you learned how to define default values for formal parameters, which made it optional to provide values for those parameters when invoking the functions.

In this chapter, you'll see one more way to invoke functions with optional parameters, with keyword-based parameter passing. This is particularly convenient when there are several optional parameters and you want to provide a value for one of the later parameters while not providing a value for the earlier ones.

The online official python documentation includes a tutorial on optional parameters which covers the topic quite well. Please read the content there: Keyword arguments

Don't worry about the `def cheeseshop(kind, *arguments, **keywords):` example. You should be able to get by without understanding `*parameters` and `**parameters` in this course. But do make sure you understand the stuff above that.

The basic idea of passing arguments by keyword is very simple. When invoking a function, inside the parentheses there are always 0 or more values, separated by commas. With keyword arguments, some of the values can be of the form `paramname = <expr>` instead of just `<expr>`. Note that when you have `paramname = <expr>` in a function definition, it is defining the default value for a parameter when no value is provided in the invocation; when you have `paramname = <expr>` in the invocation, it is supplying a value, overriding the default for that paramname.

To make it easier to follow the details of the examples in the official python tutorial, you can step through them in CodeLens.

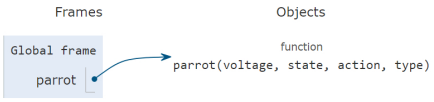
Python 3.3

```
1  def parrot(voltage, state='a stiff', action='voom', type='Norwegian Blue'):
2      print("-- This parrot wouldn't" + action,)
3      print("if you put" + str(voltage) + "volts through it.")
4      print("-- Lovely plumage, the" +  type)
5      print("-- It's " + state + "!")
6
7  parrot(1000)                                         # 1 positional argument
8  parrot(voltage=1000)                                 # 1 keyword argument
9  parrot(voltage=1000000, action='VOOOOOM')            # 2 keyword arguments
10 parrot(action='VOOOOOM', voltage=1000000)            # 2 keyword arguments
11 parrot('a million', 'bereft of life', 'jump')        # 3 positional arguments
12 parrot('a thousand', state='pushing up the daisies') # 1 positional, 1 keyword
```

<< 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:

```
-- Lovely plumage, theNorwegian Blue
-- It's a stiff!
-- This parrot wouldn'tjump
if you puta millionvolts through it.
-- Lovely plumage, theNorwegian Blue
-- It's bereft of life!
-- This parrot wouldn'tvoom
if you puta thousandvolts through it.
-- Lovely plumage, theNorwegian Blue
-- It's pushing up the daisies!
```

Activity: 1 -- CodeLens: (keyword_params_1)

As you step through it, each time the function is invoked, make a prediction about what each of the four parameter values will be during execution of lines 2-5. Then, look at the stack frame to see what they actually are during the execution.

Note

Note that we have yet another, slightly different use of the = sign here. As a stand-alone, top-level statement, `x=3` , the variable x is set to 3. Inside the parentheses that invoke a function, `x=3` says that 3 should be bound to the local variable x in the stack frame for the function invocation. Inside the parentheses of a function definition, `x=3` says that 3 should be the value for x in every invocation of the function where no value is explicitly provided for x.

15.2.1. Keyword Parameters with .format

Earlier you learned how to use the `format` method for strings, which allows you to structure strings like fill-in-the-blank sentences. Now that you've learned about optional and keyword parameters, we can introduce a new way to use the `format` method.

This other option is to specifically refer to keywords for interpolation values, like below.

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

```
1 names_scores = [("Jack",[67,89,91]),("Emily",[72,95,42]),("Taylor",[83,92,86])]
2 for name, scores in names_scores:
```

```
    print("The scores {nm} got were: {s1},{s2},{s3}.".format(nm=name,s1=scores[0],s
```

```
The scores Jack got were: 67,89,91.
The scores Emily got were: 72,95,42.
The scores Taylor got were: 83,92,86.
```

Activity: 2 -- ActiveCode (ac15_2_1)

Sometimes, you may want to use the `.format` method to insert the same value into a string multiple times. You can do this by simply passing the same string into the format method, assuming you have included `{}` s in the string everywhere you want to interpolate them. But you can also use positional passing references to do this! The order in which you pass arguments into the `format` method matters: the first one is argument `0`, the second is argument `1`, and so on.

For example,

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

```
# this works
names = ["Jack","Jill","Mary"]
for n in names:
    print("'{}!' she yelled. '{}! {}, {}!'".format(n,n,n,"say hello"))

# but this also works!
names = ["Jack","Jill","Mary"]
for n in names:
    print("'{0}!' she yelled. '{0}! {0}, {1}!'".format(n,"say hello"))

```

```
'Jack!' she yelled. 'Jack! Jack, say hello!'
'Jill!' she yelled. 'Jill! Jill, say hello!'
'Mary!' she yelled. 'Mary! Mary, say hello!'
'Jack!' she yelled. 'Jack! Jack, say hello!'
'Jill!' she yelled. 'Jill! Jill, say hello!'
'Mary!' she yelled. 'Mary! Mary, say hello!'
```

Activity: 3 -- ActiveCode (ac15_2_2)

Check your understanding

advfuncs-2-1: What value will be printed for z?

```
initial = 7
def f(x, y = 3, z = initial):
    print("x, y, z are:", x, y, z)

f(2, 5)
```

- A. 2
- B. 3
- C. 5
- D. 7
- E. Runtime error since not enough values are passed in the call to f

Check me | Compare me

✔ 2 is bound x, 5 to y, and z gets its default value, 7

Activity: 4 -- Multiple Choice (question15_2_1)

advfuncs-2-2: What value will be printed for y?

```
initial = 7
def f(x, y = 3, z = initial):
    print("x, y, z are:", x, y, z)

f(2, z = 10)
```

- A. 2
- B. 3
- C. 5
- D. 10
- E. Runtime error since no value is provided for y, which comes before z

Check me Compare me

✔ 3 is the default value for y, and no value is specified for y,

Activity: 5 -- Multiple Choice (question15_2_2)

advfuncs-2-3: What value will be printed for x?

```
initial = 7
def f(x, y = 3, z = initial):
    print("x, y, z are:", x, y, z)

f(2, x=5)
```

- A. 2
- B. 3
- C. 5
- D. 7
- E. Runtime error since two different values are provided for x

Check me Compare me

✔ 2 is bound to x since it's the first value, but so is 5, based on keyword.

Activity: 6 -- Multiple Choice (question15_2_3)

advfuncs-2-4: What value will be printed for z?

```
initial = 7
def f(x, y = 3, z = initial):
    print ("x, y, z are:", x, y, z)
initial = 0
f(2)
```

- A. 2
- B. 7
- C. 0
- D. Runtime error since two different values are provided for initial.

Check me Compare me

✔ the default value for z is determined at the time the function is defined; at that time initial has the value 0.

Activity: 7 -- Multiple Choice (question15_2_4)

advfuncs-2-5: What value will be printed below?

```
names = ["Alexey", "Catalina", "Misuki", "Pablo"]
print("'{first}!' she yelled. 'Come here, {first}! {f_one}, {f_two}, and {f_three} are here!'".format(first = names[1], f_one = names[0], f_two = names[2], f_three = names[3]))
```

- A. 'first!' she yelled. 'Come here, first! f_one, f_two, and f_three are here!'
- B. 'Alexey!' she yelled. 'Come here, Alexey! Catalina, Misuki, and Pablo are here!'
- C. 'Catalina!' she yelled. 'Come here, Catalina! Alexey, Misuki, and Pablo are here!'
- D. There is an error. You cannot repeatedly use the keyword parameters.

Check me Compare me

✔ Yes, the keyword parameters will determine the order of the strings.

Activity: 8 -- Multiple Choice (question15_2_5)

5. Define a function called `multiply`. It should have one required parameter, a string. It should also have one optional parameter, an integer, named `mult_int`, with a default value of 10. The function should return the string multiplied by the integer. (i.e.: Given inputs "Hello", mult_int=3, the function should return "HelloHelloHello")

Save & Run 5/14/2021, 4:08:49 PM - 2 of 2 Show in CodeLens

```
1
2
3 def multiply(x,mult_int=10):
4         return x*mult_int
5
```

Activity: 9 -- ActiveCode (ac15_2_3)

Result	Actual Value	Expected Value	Notes
Pass	'HelloHelloHello'	'HelloHelloHello'	Testing the function multiply on inputs 'Hello', 3.
Pass	'Goodb...odbye'	'Goodb...odbye'	Testing the function mulitply on input 'Goodbye'.

Expand Differences

You passed: 100.0% of the tests

6. Currently the function is supposed to take 1 required parameter, and 2 optional parameters, however the code doesn't work. Fix the code so that it passes the test. This should only require changing one line of code.

Save & Run 5/14/2021, 4:10:02 PM - 2 of 2 Show in CodeLens

```
1
2 def waste(mar,var = "Water",marble = "type"):
3     final_string = var + " " + marble + " " + mar
4     return final_string
5
```

Activity: 10 -- ActiveCode (ac15_2_4)

Result	Actual Value	Expected Value	Notes
Pass	'Water...kemon'	'Water...kemon'	Testing that waste returns the correct string on input 'Pokemon'

Expand Differences

You passed: 100.0% of the tests

You have attempted 11 of 10 activities on this page

15.1. Introduction: Optional Parameters">
15.3. Anonymous functions with lambda expressions">
troduction: Optional Parameters">
15.3. Anonymous functions with lambda expressions">Next Section - 15.3. Anonymous functions with lambda expressions

| Back to top