Exercise 18: Names, Variables, Code, Functions

Big title, right? I am about to introduce you to *the function*! Dum dum dah! Every programmer will go on and on about functions and all the different ideas about how they work and what they do, but I will give you the simplest explanation you can use right now.

Functions do three things:

1. They name pieces of code the way variables name strings and numbers.
2. They take arguments the way your scripts take `argv`.
3. Using 1 and 2 they let you make your own "mini-scripts" or "tiny commands."

You can create a function by using the word `def` in Python. I'm going to have you make four different functions that work like your scripts, and I'll then show you how each one is related.

```
1  # this one is like your scripts with argv
2  def print_two(*args):
3      arg1, arg2 = args
4      print "arg1: %r, arg2: %r" % (arg1, arg2)
5
6  # ok, that *args is actually pointless, we can just do this
7  def print_two_again(arg1, arg2):
8      print "arg1: %r, arg2: %r" % (arg1, arg2)
9
10 # this just takes one argument
11 def print_one(arg1):
12     print "arg1: %r" % arg1
13
14 # this one takes no arguments
15 def print_none():
16     print "I got nothin'."
17
18
19 print_two("Zed","Shaw")
20 print_two_again("Zed","Shaw")
21 print_one("First!")
22 print_none()
```

Let's break down the first function, `print_two`, which is the most similar to what you already know from making scripts:

1. First we tell Python we want to make a function using `def` for "define".
2. On the same line as `def` we give the function a name. In this case we just called it "print_two" but it could also be "peanuts". It doesn't matter, except that your function should have a short name that says what it does.
3. Then we tell it we want `*args` (asterisk args) which is a lot

like your `argv` parameter but for functions. This *has* to go inside `()` parentheses to work.
4. Then we end this line with a `:` colon, and start indenting.
5. After the colon all the lines that are indented four spaces will become attached to this name, `print_two`. Our first indented line is one that unpacks the arguments the same as with your scripts.
6. To demonstrate how it works we print these arguments out, just like we would in a script.

The problem with `print_two` is that it's not the easiest way to make a function. In Python we can skip the whole unpacking arguments and just use the names we want right inside `()`. That's what `print_two_again` does.

After that you have an example of how you make a function that takes one argument in `print_one`.

Finally you have a function that has no arguments in `print_none`.

WARNING

This is very important. Do *not* get discouraged right now if this doesn't quite make sense. We're going to do a few exercises linking functions to your scripts and show you how to make more. For now just keep thinking "mini-script" when I say "function" and keep playing with them.

What You Should See

If you run `ex18.py` you should see:

```
$ python ex18.py
arg1: 'Zed', arg2: 'Shaw'
arg1: 'Zed', arg2: 'Shaw'
arg1: 'First!'
I got nothin'.
```

Right away you can see how a function works. Notice that you used your functions the way you use things like `exists`, `open`, and other "commands." In fact, I've been tricking you because in Python those "commands" are just functions. This means you can make your own commands and use them in your scripts too.

Study Drills

Create a *function checklist* for later exercises. Write these checks on an index card and keep it by you while you complete the rest of these exercises or until you feel you do not need the index card anymore:

1. Did you start your function definition with `def`?
2. Does your function name have only characters and `_` (underscore) characters?
3. Did you put an open parenthesis `(` right after the function

name?
4. Did you put your arguments after the parenthesis `(` separated by commas?
5. Did you make each argument unique (meaning no duplicated names)?
6. Did you put a close parenthesis and a colon `):` after the arguments?
7. Did you indent all lines of code you want in the function four spaces? No more, no less.
8. Did you "end" your function by going back to writing with no indent (`dedenting` we call it)?

When you run ("use" or "call") a function, check these things:

1. Did you call/use/run this function by typing its name?
2. Did you put the `(` character after the name to run it?
3. Did you put the values you want into the parenthesis separated by commas?
4. Did you end the function call with a `)` character?

Use these two checklists on the remaining lessons until you do not need them anymore.

Finally, repeat this a few times to yourself:

```
"To 'run,' 'call,' or 'use' a function all mean the same
thing."
```

Common Student Questions

What's allowed for a function name?

The same as variable names. Anything that doesn't start with a number, and is letters, numbers, and underscores will work.

What does the `*` in `*args` do?

That tells Python to take all the arguments to the function and then put them in `args` as a list. It's like `argv` that you've been using, but for functions. It's not normally used too often unless specifically needed.

This feels really boring and monotonous.

That's good. It means you're starting to get better at typing in the code and understanding what it does. To make it less boring, take everything I tell you to type in, and then break it on purpose.

Video

- Email help from the author.
- See a list of everything you get before you buy.

When you buy the videos they will immediately show up **right here** without any hassles.

Already Paid? Reactivate Your Purchase Right Now!

Email Address

Pay With Credit Card (by Stripe™)

Use your PayPal™ account.

Buy Learn Python The Hard Way, 3rd Edition