Exercise 13: Parameters, Unpacking, Variables

In this exercise we will cover one more input method you can use to pass variables to a script (script being another name for your `.py` files). You know how you type `python ex13.py` to run the `ex13.py` file? Well the `ex13.py` part of the command is called an "argument." What we'll do now is write a script that also accepts arguments.

Type this program and I'll explain it in detail:

```
1  from sys import argv
2
3  script, first, second, third = argv
4
5  print "The script is called:", script
6  print "Your first variable is:", first
7  print "Your second variable is:", second
8  print "Your third variable is:", third
```

On line `1` we have what's called an "import." This is how you add features to your script from the Python feature set. Rather than give you all the features at once, Python asks you to say what you plan to use. This keeps your programs small, but it also acts as documentation for other programmers who read your code later.

The `argv` is the "argument variable," a very standard name in programming, that you will find used in many other languages. This variable *holds* the arguments you pass to your Python script when you run it. In the exercises you will get to play with this more and see what happens.

Line 3 "unpacks" `argv` so that, rather than holding all the arguments, it gets assigned to four variables you can work with: `script`, `first`, `second`, and `third`. This may look strange, but "unpack" is probably the best word to describe what it does. It just says, "Take whatever is in argv, unpack it, and assign it to all of these variables on the left in order."

After that we just print them out like normal.

Hold Up! Features Have Another Name

I call them "features" here (these little things you `import` to make your Python program do more) but nobody else calls them features. I just used that name because I needed to trick you into learning what they are without jargon. Before you can continue, you need to learn their real name: `modules`.

From now on we will be calling these "features" that we `import` *modules*. I'll say things like, "You want to import the `sys` module." They are also called "libraries" by other programmers, but let's just stick with modules.

What You Should See

Run the program like this (and you *must* pass *three* command line arguments):

```
$ python ex13.py first 2nd 3rd
The script is called: ex13.py
Your first variable is: first
Your second variable is: 2nd
Your third variable is: 3rd
```

This is what you should see when you do a few different runs with different arguments:

```
$ python ex13.py stuff things that
The script is called: ex13.py
Your first variable is: stuff
Your second variable is: things
Your third variable is: that
$
$ python ex13.py apple orange grapefruit
The script is called: ex13.py
Your first variable is: apple
Your second variable is: orange
Your third variable is: grapefruit
```

You can actually replace `first`, `2nd`, and `3rd` with any three things you want.

If you do not run it correctly, then you will get an error like this:

```
$ python ex13.py first 2nd
Traceback (most recent call last):
  File "ex13.py", line 3, in <module>
    script, first, second, third = argv
ValueError: need more than 3 values to unpack
```

This happens when you do not put enough arguments on the command when you run it (in this case just `first 2nd`). Notice when I run it I give it `first 2nd`, which caused it to give an error about "need more than 3 values to unpack" telling you that you didn't give it enough parameters.

Study Drills

1. Try giving fewer than three arguments to your script. See that error you get? See if you can explain it.
2. Write a script that has fewer arguments and one that has more. Make sure you give the unpacked variables good names.
3. Combine `raw_input` with `argv` to make a script that gets more input from a user.
4. Remember that modules give you features. Modules. Modules. Remember this because we'll need it later.

Common Student Questions

When I run it I get `ValueError: need more than 1 value to unpack`.

Remember that an important skill is paying attention to details. If you look at the *What You Should See* section you see that I run the script with parameters on the command line. You should replicate how I ran it exactly.

What's the difference between `argv` and `raw_input()`?

The different has to do with where the user is required to give input. If they give your script inputs on the command line, then you use `argv`. If you want them to input using the keyboard while the script is running, then use `raw_input()`.

Are the command line arguments strings?

Yes, they come in as strings, even if you typed numbers on the command line. Use `int()` to convert them just like with `raw_input()`.

How do you use the command line?

You should have learned to use it real quick by now, but if you need to learn it at this stage, then read the Command Line Crash Course I wrote for this book in Appendix A.

I can't combine `argv` with `raw_input()`.

Don't overthink it. Just slap two lines at the end of this script that uses `raw_input()` to get something and then print it. From that start playing with more ways to use both in the same script.

Why can't I do this `raw_input('? ') = x`?

Because that's backward to how it should work. Do it the way I do it and it'll work.

Email Address

Buy Learn Python The Hard Way, 3rd Edition

If you've bought something before we'll have a record of it. Otherwise we'll make you a new account.

 (/book/)

 (mailto:help@learncodethehardway.org)