Theory: Command line arguments

19 minutes 0 / 5 problems solved

§1. Running from the command line

Using the command line is sometimes very useful in the programmer's work. And python scripts can be run from the command line just like its regular commands, e.g. "cd" or "mkdir". This means we can write a module that, for example, multiplies two numbers and nicely prints the result, and run it from the shell:

```
python multiply_two_numbers.py 5 9
```

In the line above, `python` is kind of a command that indicates that the python interpreter should be used for the following script. In some cases, the system may already know how to run *.py* files but we will not go into details here and, for the sake of consistency, will use the `python` command throughout this topic.

Then, separated by a whitespace, follows the script name. Note that if the script is in another directory than you are working from, you should specify the path to the file. It may be an absolute path:

```
python C:\python_scripts\add_two_numbers.py 11 44
```

Or it can be a relative path, for example to run a script from the parent directory:

```
python ..\add_two_numbers.py 11 44
```

Finally, if the script takes any arguments, they are written separated by whitespaces after the script name.

And that's it! However, the next question is – how can we get access to the specified arguments from our python script?

§2. System module

In order to do so, we can make use of the **sys module**. It provides access to functions and variables that allow for working with the underlying Python interpreter, irrespective of the operating system. We won't go into details talking about its features, but rather focus on the one that is the most important right now, namely, **sys.argv**. It performs the very operation we need: collects the arguments passed to the python script.

By calling `sys.argv`, we get arguments specified by the user as **a list of strings**. Indexing, as always in Python, starts from *0* but the first argument, `sys.argv[0]` is the name of our python script as it was invoked – either the name itself or including the path to the file. And the next list items are arguments that can also be accessed by their index. Take note that they are strings, and if we need a numerical value, we should perform type conversion.

Let's write a simple program `multiply_two_numbers.py`:

```
import sys  # first, we import the module

args = sys.argv  # we get the list of arguments
first_num = float(args[1])  # convert arguments to float
second_num = float(args[2])

product = first_num * second_num


print("The product of " + args[1] + " times " + args[2] + " equals " + str(product
))
```

Current topic:

Command line arguments

Topic depends on:

✓ Type casting 12★

✓ Indexes 7★ Stage 3

✓ Load module 5★ Stage 1

✗ Parameters and options Stage 1

Topic is required for:

Argparse module

Table of contents:

↑ Command line arguments

§1. Running from the command line

§2. System module

§3. Checking the input

§4. Within the IDE

§5. Conclusion

Feedback & Comments

§3. Checking the input

It is also worth mentioning that if we expect to get a specific number of arguments (i. e. almost always), it is a good idea to check the length of `sys.argv` in the program. Let's check that in our script `multiply_two_numbers.py`:

```
import sys

args = sys.argv

if len(args) != 3:

 print("The script should be called with two arguments, the first and the secon
d number to be multiplied")

else:
    first_num = float(args[1])

    second_num = float(args[2])



    product = first_num * second_num



 print("The product of " + args[1] + " times " + args[2] + " equals " + str(pro
duct))
```

So, this is how our script will look like from the command line:

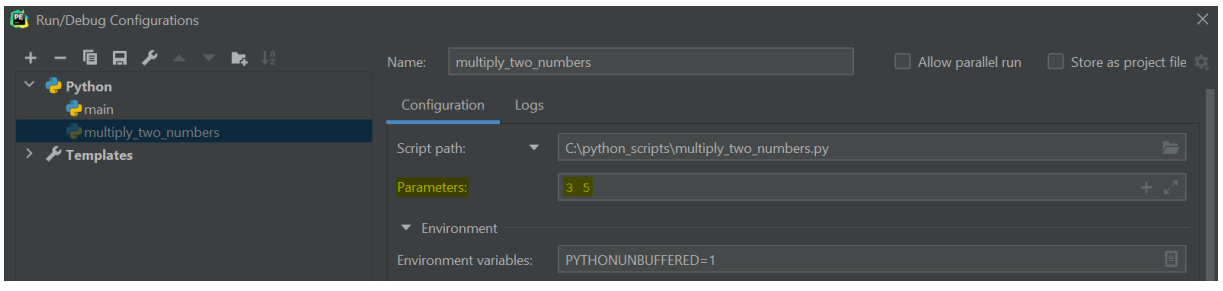
```
C:\>python multiply_two_numbers.py 75 3
The product of 75 times 3 equals 225.0

C:\>python multiply_two_numbers.py
The script should be called with two arguments, the first and the second number to be multiplied

C:\>
```

§4. Within the IDE

Let's take a look at **PyCharm**'s capabilities in comparison to the command line. Instead of manually writing the script name and arguments each time, you can set them in the configuration. For this in the **Run** area select **Edit Configurations** to open the *Run/Debug Configurations* dialog.

If you do not see a similar area in your IDE, then make it visible through **View -> Appearance -> Navigation Bar**. You can read more on how to do it in the JetBrains documentation.

Congratulations, you got into the *Run/Debug Configurations*! In the **Parameters** field, we can set the arguments that we would write in the command line separated by whitespaces.



Save changes and run the script. The output would be as expected:

```
The product of 3 times 5 equals 15.0
```

Now, instead of running the module from the shell as `python multiply_two_numbers.py 3 5` and passing arguments each time it is called, you can set them in the **Parameters** field and just run the program in **PyCharm**.

§5. Conclusion

We have learned how to run python scripts from the command line, how to get access to the passed arguments from the program itself, as well as that it's important to check that the arguments are what we expect them to be. We also got acquainted with **PyCharm**'s capabilities for specifying script arguments in configurations. This knowledge will definitely help you in your further programmer's path!

Report a typo

363 users liked this theory. **19** didn't like it. **What about you?**

Start practicing

Comments (17) Hints (3) Useful links (2) Show discussion