Theory: Experiments with Python shell

12 minutes 0 / 5 problems solved

You already know how to start the Python shell and run some simple code there. Let's see how to use it for experiments. We will illustrate all of them with IDLE samples.

For instance, if you want to study a new module or library, you can conduct some code experiments in the Python shell to understand how to work with it. You can also implement your functions and classes in the shell and test them before including in the final script.

§1. Methods, modules, and functions

Imagine you've just heard about the method `type()` and want to learn more about it. You can simply start the shell and, using your own examples, see how it works:

```
>>> a = 'this is a string'
>>> b = 4
>>> c = 7.0
>>> type(a)
<class 'str'>
>>> type(b)
<class 'int'>
>>> type(c)
<class 'float'>
```

The same can be done with modules:

```
>>> import math
>>> math.sqrt(4)
2.0
>>>
```

The `math` module contains tools for mathematic calculations. In this example, we have imported it and tested its function that returns the square root of the given value.

This function works so that if we pass a non-numerical value as an argument, e.g. a string, it will return a TypeError.

```
>>> math.sqrt('5')
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    math.sqrt('5')
TypeError: must be real number, not str
>>>
```

If we don't want the program to stop executing in such cases, we can alter the method ourselves. That's how we can implement and use our own functions in the shell:

```
>>> import math
>>> def my_sqrt(value):
        if type(value) is not int and type(value) is not float:
            return None
        else:
            return(math.sqrt(value))


>>> my_sqrt(8)
2.8284271247461903
>>> my_sqrt('8')
>>>
```

This is a simple modification of the `math.sqrt()` function that returns `None` if the passed value was neither an integer nor a float. So, in this experiment, you've found out how to implement your own function in the shell, learned how to use the `type()` method and got to know the `sqrt()` function of the `math` module.

Remember: everything you do in IDLE or command line shell disappears once you close it; you won't be able to use this code afterward. On the other hand, it's good because you definitely won't break anything in the program

Current topic:

Experiments with Python shell — Stage 2

Topic depends on:

- ✕ Introduction to Python shell — Stage 2
- ✓ Declaring a function 10★ — Stage 1
- ✓ Load module 5★ — Stage 1

Topic is required for:

- Debugging in shell — Stage 2

Table of contents:

↑ Experiments with Python shell

§1. Methods, modules, and functions

§2. Magic? Autocomplete!

§3. Recap

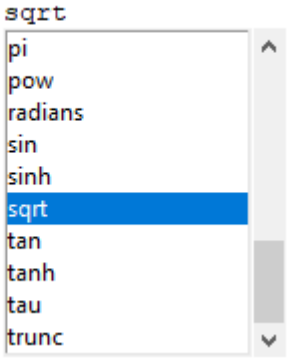
Feedback & Comments

that you have copied in the shell and work with it there.

§2. Magic? Autocomplete!

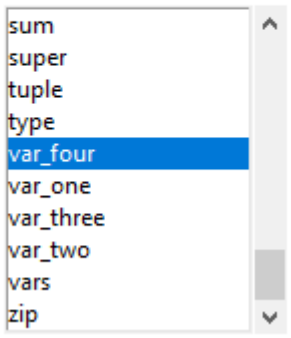
The Python shell also has an autocomplete function which is very useful when you're studying new things. In case that you don't remember the exact name of the method you'd like to use or even have no idea how it is called or you want to look up which methods are in the module, type a module name and a dot, then wait for a couple of seconds — the shell will help you with the list of available methods.

```
>>> import math
>>> math.sqrt
```



The next beautiful thing there is **autocomplete** for names of functions and variables. Use the `TAB` key for the shell to offer the function or variable name for you. You just need to type first letters and press `TAB`. In case of several names, you'll see a list of them:

```
>>> var_one = "this"
>>> var_two = "is"
>>> var_three = "an"
>>> var_four = "example"
>>> var
```



This won't work if you launch the shell from the command line interpreter.

§3. Recap

The Python shell provides a perfect opportunity both to play around with new modules and to refresh your memory about familiar ones. You can quickly test how methods and functions work, look up which methods an object has or which functions there are in any module. And such a useful option as autocomplete can help you with that!

Report a typo

128 users liked this theory. **2** didn't like it. **What about you?**

Start practicing

Comments (2) Hints (0) Useful links (1) Show discussion