Theory: Load module

16 minutes 10 / 10 problems solved

Start practicing

8974 users solved this topic. Latest completion was about 1 hour ago.

§1. Module basics

While working on simple examples you probably type your code directly into the interpreter. But every time you quit from the interpreter and start it again you lose all the definitions you made before. So as you start writing larger programs it makes sense to prepare your code in advance using a text editor and then run it with the interpreter. A file containing a list of operations that further are read and interpreted is called **script**.

You also may want to write some functions and then use them in other programs or even reuse code someone else wrote before. One way is just to copy the code into your program, but it soon leads to code that is bad-structured and hard to read. Luckily, there is another way in Python to organize and reuse code called **modules**.

The module is simply a file that contains Python statements and definitions. It usually has a **.py** extension. What really makes the module system powerful is the ability to **load** or **import** one module from another.

§2. Module loading

To load a module just use an **import** statement. In a basic form, it has the following syntax `import module`.

```
import super_module

super_module.super_function()  # calling a function defined in super_module

print(super_module.super_variable)  # accessing a variable defined in super_module
```

`super_module` is the name of the module you want to import. For example, a file called **`super_module.py`** has a name **`super_module`**. In order to be available for import, **`super_module.py`** should be located in the same directory as the file you are trying to import it from. At first, Python importing system looks for a module in the current directory, then it checks the built-in modules, and if nothing is found an error will be raised. After importing, the module becomes available under its name and you can access functions and variables defined in it using the dot notation.

It's also common to only import required functions or variables from a module but not the module itself. You can do this by using a **from** form of import statement.

```
from super_module import super_function

super_function()  # super_function is now available directly at the current module

super_module.super_function()  # note, that in this case name super_module is not imported,
                               # so this line leads to an error
```

A good practice is to load a single module in a single line and put all your imports at the top of the file because it increases readability.

```
import module1
import module2
import module3

# the rest of module code goes here
```

Current topic:

✓ Load module (Stage 1, 5★)

Topic depends on:

✓ Invoking a function (Stage 1, 16★)

Topic is required for:

✓ Create module (Stage 1)
✓ Defaultdict and Counter
Copy of an object
Experiments with Python shell (Stage 2)
Math functions
✓ Random module (Stage 4, 4★)
✓ Stack in Python
✓ Queue in Python
Datetime module (Stage 2)
Time module
✓ Itertools module
Socket module
Json module (Stage 2)
Regexps in Python
os module
Unit testing in Python
Command line arguments

Table of contents:

A special form of import statement allows you to load all the names defined in a module. It is called *wildcard import* and has syntax `from module import *`. You should generally avoid this in your code. It can cause unexpected behavior because you don't know what names exactly are imported into the current namespace. Besides, these names may shadow some of the existing ones without your knowledge. It's better to make it explicit and specify what you're importing.

In case you have to use several import statements, pay attention to their order:

1. standard library imports
2. third party dependency imports
3. local application imports

Having your imports grouped, you may put a blank line between import sections. Also, some guidelines, including ours, recommend sorting imports alphabetically.

§3. Built-in modules

Python comes with a great standard library. It contains a lot of built-in modules that provide useful functions and data structures. Another advantage is that the standard library is available on every system that has Python installed. Here you can find an official library reference.

Python has a `math` module that provides access to mathematical functions.

```
import math

print(math.factorial(5))  # prints the value of 5!

print(math.log(10))  # prints the natural logarithm of 10

print(math.pi)  # math also contains several constants
print(math.e)
```

`string` module contains common string operations and constants.

```
from string import digits

print(digits)  # prints all the digit symbols
```

`random` module provides functions that let you make a random choice.

```
from random import choice


print(choice(['red', 'green', 'yellow']))  # print a random item from the list
```

Report a typo

Start practicing

Comments (8)   Hints (0)   Useful links (0)   Show discussion