AskPython

Python import Statement

By Safa Mulani / January 7, 2020

Python `import` statement enables the user to import particular modules in the corresponding program.

It resembles the #include header_file in C/C++.

As soon as the interpreter encounters the import statement in a particular code, it searches for the same in the local scope and imports the module, if present in the search path.

It searches for a particular module in its built-in modules section at first. If it's not found, it searches those modules in its current directory.

A module is loaded only once in a particular program, without being affected by the number of times the module is imported.

Syntax:

```
import module_name
```

Example:

```python
import collections
```

1. Importing class/functions from a module

We can import classes/functions from a module using the syntax:

```
from {module} import {class/function}
```

Example:

```
from collections import OrderedDict
from os import path
from math import pi
print(pi)
```

Output:

```
3.141592653589793
```

2. The *import* * Statement

All the methods and constants of a particular module can be imported using import * operator.

```
from math import *
print(pi)
print(floor(3.15))
```

Output:

```
3.141592653589793
3
```

3. Python’s *import as* Statement

The `import as` statement helps the user provide an alias name to the original module name.

```
# python import as
import math as M

print(M.pi)
print(M.floor(3.18))
```

Output:

```
3.141592653589793
3
```

4. Importing user-defined modules

We can import the functions of one program into another using its name.

Initially, we need to create a python code.

test.py

```
def sub(a, b):
    return int(a) - int(b)

def lower_case(str1):
    return str(str1).lower()
```

Then create another python script, wherein we need to import the above create test.py script.

test2.py

```
import test

print(test.sub(5,4))
print(test.lower_case('SafA'))
```

Output:

```
1
safa
```

5. Importing from another directory

The `importlib` library is used to import a script from another directory.

Initially, we need to create a python script and define functions in it.

test1.py

```
def sub(a, b):
    return int(a) - int(b)

def lower_case(str1):
    return str(str1).lower()
```

Then, we will create another python script and save it into another directory and then import the functionalities from test1.py (which resides into another directory).

design.py

```
import importlib, importlib.util

def module_directory(name_module, path):
    P = importlib.util.spec_from_file_location(name_module, path)
    import_module = importlib.util.module_from_spec(P)
    P.loader.exec_module(import_module)
    return import_module

result = module_directory("result", "../inspect_module/test1.py")

print(result.sub(3,2))
print(result.lower_case('SaFa'))
```

Output:

```
1
safa
```

Another alternative way is to add the module directory to the **sys.path** list.

6. Importing class from another file

tests.py

```
class Employee:
    designation = ""

    def __init__(self, result):
        self.designation = result

    def show_designation(self):
        print(self.designation)


class Details(Employee):
    id = 0

    def __init__(self, ID, name):
        Employee.__init__(self, name)
        self.id = name

    def get_Id(self):
        return self.id
```

design.py

```
import importlib, importlib.util

def module_directory(name_module, path):
    P = importlib.util.spec_from_file_location(name_module, path)
    import_module = importlib.util.module_from_spec(P)
    P.loader.exec_module(import_module)
    return import_module

result = module_directory("result", "../Hello/tests.py")

a = result.Employee('Project Manager')
a.show_designation()

x = result.Details(4001,'Safa')
x.show_designation()
print(x.get_Id())
```

Output:

```
Project Manager
Safa
Safa
```

Conclusion

Thus, in this article, we have understood the functionalities offered by the import statement.

References

- Python import statement
- import statement documentation

← Previous Post

Next Post →

Search...

Recent Posts

Connected Health: Exploring IoT Solutions for Healthcare Transformation

Maximizing Cost Savings Through Offshore Development: A Comprehensive Guide

Running a Flask Application with Python3: A Step-by-Step Guide

Levenshtein Distance in Python: Troubleshooting Installation Errors on Windows

10 PyJanitor's Miscellaneous Functions for Enhancing Data Cleaning

Random Search in Machine Learning: Hyperparameter Tuning Technique

Flake8: Python's Powerful Code Analysis Tool for Improved Code Quality

Delivery Route Optimization using Python: A Step-by-Step Guide

Understanding Capital Asset Pricing Model (CAPM)

Splitting Lists into Sub-Lists in Python: Techniques and Advantages

Favorite Sites

GoLang Tutorials

VM-Help

Linux Tutorials

MySQL Tutorials

CodeForGeek

Mkyong