Python Modules

June 9, 2024

Dr.Labeed Al-Saad

0.1 What is a Module?

Consider a module to be the same as a code library.

A file containing a set of functions you want to include in your application.

0.2 Create a Module

To create a module just save the code you want in a file with the file extension .py:

Example:

Save this code in a file named mymodule.py

[ ]:
```
def greeting(name):
  print("Hello, " + name)
```

0.3 Use a Module

Now we can use the module we just created, by using the import statement:

Example:

Import the module named mymodule, and call the greeting function:

[ ]:
```
import mymodule

mymodule.greeting("Jonathan")
```

**Note: When using a function from a module, use the syntax: module_name.function_name.

0.4 Variables in Module

The module can contain functions, as already described, but also variables of all types (arrays, dictionaries, objects etc):

Example:

Save this code in the file mymodule.py

[ ]:
```
person1 = {
  "name": "John",
  "age": 36,
  "country": "Norway"
}
```

Example:

Import the module named mymodule, and access the person1 dictionary:

[ ]:
```
import mymodule

a = mymodule.person1["age"]
print(a)
```

36

0.5 Naming a Module

You can name the module file whatever you like, but it must have the file extension .py

Re-naming a Module You can create an alias when you import a module, by using the as keyword:

Example:

Create an alias for mymodule called mx:

[ ]:
```
import mymodule as mx

a = mx.person1["age"]
print(a)
```

36

0.6 Built-in Modules

There are several built-in modules in Python, which you can import whenever you like.

Example:

Import and use the platform module:

[6]:
```
import platform

x = platform.system()
print(x)
```

```
Windows
```

0.7 Using the dir() Function

There is a built-in function to list all the function names (or variable names) in a module. The dir() function:

Example:

List all the defined names belonging to the platform module:

[7]:
```
import platform

x = dir(platform)
print(x)
```

```
['_Processor', '_WIN32_CLIENT_RELEASES', '_WIN32_SERVER_RELEASES',
'__builtins__', '__cached__', '__copyright__', '__doc__', '__file__',
'__loader__', '__name__', '__package__', '__spec__', '__version__',
'_comparable_version', '_component_re', '_default_architecture',
'_follow_symlinks', '_get_machine_win32', '_ironpython26_sys_version_parser',
'_ironpython_sys_version_parser', '_java_getprop', '_libc_search',
'_mac_ver_xml', '_node', '_norm_version', '_os_release_cache',
'_os_release_candidates', '_os_release_line', '_os_release_unescape',
'_parse_os_release', '_platform', '_platform_cache', '_pypy_sys_version_parser',
'_sys_version', '_sys_version_cache', '_sys_version_parser', '_syscmd_file',
'_syscmd_ver', '_uname_cache', '_unknown_as_blank', '_ver_output',
'_ver_stages', 'architecture', 'collections', 'freedesktop_os_release',
'functools', 'itertools', 'java_ver', 'libc_ver', 'mac_ver', 'machine', 'node',
'os', 'platform', 'processor', 'python_branch', 'python_build',
'python_compiler', 'python_implementation', 'python_revision', 'python_version',
'python_version_tuple', 're', 'release', 'sys', 'system', 'system_alias',
'uname', 'uname_result', 'version', 'win32_edition', 'win32_is_iot',
'win32_ver']
```

**Note: The dir() function can be used on all modules, also the ones you create yourself.

0.8 Import From Module

You can choose to import only parts from a module, by using the from keyword.

Example:

The module named mymodule has one function and one dictionary:

[ ]:
```
def greeting(name):
  print("Hello, " + name)

person1 = {
  "name": "John",
  "age": 36,
  "country": "Norway"
}
```

[ ]:
```
Example:

Import only the person1 dictionary from the module:
```

[ ]:
```
from mymodule import person1

print (person1["age"])
```

36

**Note: When importing using the from keyword, do not use the module name when referring to elements in the module.

Example: person1[“age”], not mymodule.person1[“age”]