

Python

Modules

Class X

Lab 12

**Lab Objectives:**

* Module
* Usage of module

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.

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) |

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") |

|  |
| --- |
| Hello, Jonathan |

Note: When using a function from a module, use the syntax: module\_name.function\_name.

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) |

Check the results!

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) |

Check results! it’s 36

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:

|  |
| --- |
| import platform  x = platform.system()  print(x) |

|  |
| --- |
| Windows |

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:

|  |
| --- |
| import platform  x = dir(platform) print(x) |

|  |
| --- |
| ['DEV\_NULL', '\_UNIXCONFDIR', 'WIN32\_CLIENT\_RELEASES', 'WIN32\_SERVER\_RELEASES', '\_\_builtins\_\_', '\_\_cached\_\_', '\_\_copyright\_\_', '\_\_doc\_\_', '\_\_file\_\_', '\_\_loader\_\_', '\_\_name\_\_', '\_\_package \_\_', '\_\_spec\_\_', '\_\_version\_\_', '\_default\_architecture', '\_dist\_try\_harder', '\_follow\_symlinks', '\_ironpython26\_sys\_version\_parser', '\_ironpython\_sys\_version\_parser', '\_java\_getprop', '\_libc\_search', '\_linux\_distribution', '\_lsb\_release\_version', '\_mac\_ver\_xml', '\_node', '\_norm\_version', '\_perse\_release\_file', '\_platform', '\_platform\_cache', '\_pypy\_sys\_version\_parser', '\_release\_filename', '\_release\_version', '\_supported\_dists', '\_sys\_version', '\_sys\_version\_cache', '\_sys\_version\_parser', '\_syscmd\_file', '\_syscmd\_uname', '\_syscmd\_ver', '\_uname\_cache', '\_ver\_output', 'architecture', 'collections', 'dist', 'java\_ver', 'libc\_ver', 'linux\_distribution', 'mac\_ver', 'machine', 'node', 'os', 'platform', 'popen', 'processor', 'python\_branch', 'python\_build', 'python\_compiler', 'python\_implementation', 'python\_revision', 'python\_version', 'python\_version\_tuple', 're', 'release', 'subprocess', 'sys', 'system', 'system\_aliases', 'uname', 'uname\_result', 'version', 'warnings', 'win32\_ver'] |

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

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"]) |