Module (1A)

Young Won Lim
12/21/23

Python 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.

https://www.w3schools.com/python/python_modules.asp

Creating a module

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

Save this code in a file named **mymodule.py**

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

https://www.w3schools.com/python/python_modules.asp

Variables in a module (1)

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

Save this code in the file **mymodule.py**

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

https://www.w3schools.com/python/python_modules.asp

Variables in a module (1)

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

Save this code in the file **mymodule.py**

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

https://www.w3schools.com/python/python_modules.asp

Variables in a module (2)

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

```
import mymodule

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

https://www.w3schools.com/python/python_modules.asp

Naming and renaming a module

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

You can create an alias when you import a module,
by using the **as** keyword:

Create an alias for **mymodule** called **mx**:

```
import mymodule as mx

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

https://www.w3schools.com/python/python_modules.asp

Built-in modules

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

Import and use the platform module:

```
import platform

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

https://www.w3schools.com/python/python_modules.asp

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 can be used on all modules,
also the ones you create yourself.

The dir() function:

List all the defined names belonging
to the platform module:

```
import platform

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

https://www.w3schools.com/python/python_modules.asp

Import From Module (1)

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

The module named **mymodule**
has one function and one dictionary:

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

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

https://www.w3schools.com/python/python_modules.asp

Import From Module (2)

Import only the person1 dictionary from the module:

```
from mymodule import person1

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

When importing using the **from** keyword,
do <u>not</u> use the module name
when referring to <u>elements</u> in the module.

Example:

person1["age"]

not
mymodule.person1["age"]

https://www.w3schools.com/python/python_modules.asp

More on Modules (1)

Usually, modules contain functions or classes,
but there can be "plain" statements in them as well.

These statements can be used to *initialize* the module.
They are only *executed* when the module is *imported*.

a module, which only consists of just one statement:

```
print("The module is imported now!")
```

```
The module is imported now!
```

We save with the name "**one_time.py**" and
import it *two times* in an interactive session:

```
import one_time
import one_time
```

```
The module is imported now!
```

https://www.w3schools.com/python/python_modules.asp

More on Modules (2)

only imported *once*.

Each module can only be imported *once*
per interpreter session or in a program or script.

If you change a module and if you want to reload it,
you must restart the interpreter again.

In Python 2.x, it was possible to reimport the module
by using the built-in reload, i.e.reload(modulename):

```
$ python
Python 2.6.5 (r265:79063, Apr 16 2010, 13:57:41)
[GCC 4.4.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
import one_time
The module is imported now!
reload(one_time)
The module is imported now!
```

https://python-course.eu/python-tutorial/modules-and-modular-programming.php

More on Modules (3-1)

This is not possible anymore in Python 3.x.
You will cause the following error:

```
import one_time

reload(one_time)

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-7-102e1bec2702> in <module>
----> 1 reload(one_time)
NameError: name 'reload' is not defined
```

https://python-course.eu/python-tutorial/modules-and-modular-programming.php

More on Modules (3-2)

Since Python 3.0 the **reload** built-in function has been moved into the imp standard library module.

So it's still possible to reload files as before, but the functionality has to be imported.

```
import imp
imp.reload(my_module).
```

Alternatively

```
from imp import reload
reload(my_module)
```

https://python-course.eu/python-tutorial/modules-and-modular-programming.php

More on Modules (4)

Example with reloading the **Python3** way:

```
$ python3
Python 3.1.2 (r312:79147, Sep 27 2010, 09:57:50)
[GCC 4.4.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
from imp import reload
import one_time
The module is imported now!
reload(one_time)
The module is imported now!
```

Since version 3.4 you should use the "**importlib**" module,
because **imp.reload** is marked as deprecated:

```
from implib import reload
import one_time
The module is imported now!
reload(one_time)
The module is imported now!
```

https://python-course.eu/python-tutorial/modules-and-modular-programming.php

Importing names form a module directly

Names from a module can directly be imported
into the importing module's symbol table:

```
from fibonacci import fib, ifib
ifib(500)
```

```
139423224561697880139724382870407283950070256587697307264108962948325571622863290691557658876222521294125
```

This does not introduce the module name
from which the imports are taken in the local symbol table.

```
import fibonacci
fibonacci.ifib(500)
```

```
from fibonacci import ifib
ifib(500)
```

It's possible but not recommended
to import *all* names defined in a module,
except those beginning with an underscore "_":

```
from fibonacci import *
fib(500)
```

This shouldn't be done in scripts but it's possible to use it in interactive sessions to save typing.

https://python-course.eu/python-tutorial/modules-and-modular-programming.php

Executing modules as scripts (1)

Essentially a Python module is a script,
so it can be run as a script:

```
$ python fibo.py            # run as a script
```

The module which has been started as a script
will be executed as if it had been imported,
but with one exception:

The system variable **name** is set to "main".

So it's possible to program different behaviour
into a module for the two cases.

https://python-course.eu/python-tutorial/modules-and-modular-programming.php

Executing modules as scripts (2)

With the following conditional statement
the file can be used as a module or as a script,

but only if it is run as a script the method **fib**
will be started with a command line argument:

```
if __name__ == "__main__":          # as a script
    import sys                      # for command line arg
    fib(int(sys.argv[1]))           # execute fib
```

***** If it is run as a script, we get the following output:

```
$ python fibo.py 50                 # run as a script
1 1 2 3 5 8 13 21 34
```

***** If it is imported as a module,
the code in the if block will not be executed:

```
import fibo
```

https://python-course.eu/python-tutorial/modules-and-modular-programming.php

Renaming a namespace

While importing a module,
the **name** of the **namespace** can be changed:

```
import math as mathematics
print(mathematics.cos(mathematics.pi))
```

```
-1.0
```

After this import, there exists a namespace **mathematics**
but no namespace **math**.

https://python-course.eu/python-tutorial/modules-and-modular-programming.php

Renaming a namespace

It's possible to import just a few methods from a module:

```
from math import pi, pow as power, sin as sinus
power(2,3)
8.0

sinus(pi)
1.2246467991473532e-16
```

https://python-course.eu/python-tutorial/modules-and-modular-programming.php

Kinds of modules

There are different kind of modules:

those written in Python
they have the suffix: **.py**

dynamically linked C modules
suffixes are: **.dll**, **.pyd**, **.so**, **.sl**, …

C-Modules linked with the Interpreter
It's possible to get a complete list of these modules:

```
import sys
print(sys.builtin_module_names)
```

```
('_abc', '_ast', '_bisect', '_blake2', '_codecs', '_codecs_cn', '_codecs_hk', '_codecs_iso2022',
'_codecs_jp', '_codecs_kr', '_codecs_tw', '_collections', '_contextvars', '_csv', '_datetime', '_functools',
'_heapq', '_imp', '_io', '_json', '_locale', '_lsprof', '_md5', '_multibytecodec', '_opcode', '_operator',
'_pickle', '_random', '_sha1', '_sha256', '_sha3', '_sha512', '_signal', '_sre', '_stat', '_string', '_struct',
'_symtable', '_thread', '_tracemalloc', '_warnings', '_weakref', '_winapi', 'array', 'atexit', 'audioop',
'binascii', 'builtins', 'cmath', 'errno', 'faulthandler', 'gc', 'itertools', 'marshal', 'math', 'mmap', 'msvcrt', 'nt',
'parser', 'sys', 'time', 'winreg', 'xxsubtype', 'zipimport', 'zlib')
```

An error message is returned for Built-in-Modules.

https://python-course.eu/python-tutorial/modules-and-modular-programming.php

Module Search Path (1-1)

If you import a module, let's say "**import** xyz",
the interpreter searches for this module
in the following locations and in the order given:

- The directory of the top-level file,
  i.e. the file being executed.
- The directories of PYTHONPATH,
  if this global environment variable
  of your operating system is set.
- standard installation path Linux/Unix e.g.
  in /usr/lib/python3.5.

https://python-course.eu/python-tutorial/modules-and-modular-programming.php

Module Search Path (1-1)

It's possible to find out
where a module is located
after it has been imported:

```
import numpy
numpy.file
'/usr/lib/python3/dist-packages/numpy/init.py'

import random
random.file
'/usr/lib/python3.5/random.py'
```

https://python-course.eu/python-tutorial/modules-and-modular-programming.php

Module Search Path (2)

The **file** attribute doesn't always exist.

This is the case with modules which are statically linked C libraries.

```
import math
math.__file__
```

OUTPUT:

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-4-bb98ec32d2a8> in <module>
      1 import math
----> 2 math.__file__
AttributeError: module 'math' has no attribute '__file__'
```

https://python-course.eu/python-tutorial/modules-and-modular-programming.php

Content of a module (1)

With the built-in function **dir**() and
the name of the module as an argument,
you can list all valid attributes and methods for that module.

```
import math
dir(math)
```

OUTPUT:

```
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan',
'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial',
'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma',
'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'pi', 'pow', 'radians', 'remainder', 'sin', 'sinh', 'sqrt', 'tan', 'tanh',
'tau', 'trunc']
```

https://python-course.eu/python-tutorial/modules-and-modular-programming.php

Content of a module (2)

Calling **dir()** without an argument,
a list with the names in the current local scope is returned:

```
import math
cities = ["New York", "Toronto", "Berlin", "Washington", "Amsterdam", "Hamburg"]
dir()
```

```
['In, 'Out, '_, '_1, '__, '___, '__builtin__, '__builtins__, '__doc__, '__loader__, '__name__, '__package__,
'__spec__, '_dh, '_i, '_i1, '_i2, '_ih, '_ii, '_iii, '_oh, 'builtins, 'cities, 'exit, 'get_ipython, 'math, 'quit']
```

https://python-course.eu/python-tutorial/modules-and-modular-programming.php

Content of a module (3)

It's possible to get a list of
the built-in functions, exceptions, and other objects
by importing the builtins module:

```
import builtins
dir(builtins)
```

OUTPUT:

```
['ArithmeticError, 'AssertionError', 'AttributeError', 'BaseException', 'BlockingIOError', 'BrokenPipeError', 'BufferError',
'BytesWarning', 'ChildProcessError', 'ConnectionAbortedError', 'ConnectionError', 'ConnectionRefusedError',
'ConnectionResetError', 'DeprecationWarning', 'EOFError', 'Ellipsis', 'EnvironmentError', 'Exception', 'False',
'FileExistsError', 'FileNotFoundError', 'FloatingPointError', 'FutureWarning', 'GeneratorExit', 'IOError', 'ImportError',
'ImportWarning', 'IndentationError', 'IndexError', 'InterruptedError', 'IsADirectoryError', 'KeyError', 'KeyboardInterrupt',
'LookupError', 'MemoryError', 'ModuleNotFoundError', 'NameError', 'None', 'NotADirectoryError', 'NotImplemented',
'NotImplementedError', 'OSError', 'OverflowError', 'PendingDeprecationWarning', 'PermissionError',
'ProcessLookupError', 'RecursionError', 'ReferenceError', 'ResourceWarning', 'RuntimeError', 'RuntimeWarning',
'StopAsyncIteration', 'StopIteration', 'SyntaxError', 'SyntaxWarning', 'SystemError', 'SystemExit', 'TabError',
'TimeoutError', 'True', 'TypeError', 'UnboundLocalError', 'UnicodeDecodeError', 'UnicodeEncodeError', 'UnicodeError',
'UnicodeTranslateError', 'UnicodeWarning', 'UserWarning', 'ValueError', 'Warning', 'WindowsError', 'ZeroDivisionError',
'__IPYTHON__', '__build_class__', '__debug__', '__doc__', '__import__', '__loader__', '__name__', '__package__',
'__spec__', 'abs', 'all', 'any', 'ascii', 'bin', 'bool', 'breakpoint', 'bytearray', 'bytes', 'callable', 'chr', 'classmethod', 'compile',
'complex', 'copyright', 'credits', 'delattr', 'dict', 'dir', 'display', 'divmod', 'enumerate', 'eval', 'exec', 'filter', 'float', 'format',
'frozenset', 'get_ipython', 'getattr', 'globals', 'hasattr', 'hash', 'help', 'hex', 'id', 'input', 'int', 'isinstance', 'issubclass', 'iter', 'len',
'license', 'list', 'locals', 'map', 'max', 'memoryview', 'min', 'next', 'object', 'oct', 'open', 'ord', 'pow', 'print', 'property', 'range',
'repr', 'reversed', 'round', 'set', 'setattr', 'slice', 'sorted', 'staticmethod', 'str', 'sum', 'super', 'tuple', 'type', 'vars', 'zip']
```

https://python-course.eu/python-tutorial/modules-and-modular-programming.php

Packages

If you have created a lot of modules at some point in time,
you may loose the overview about them.

You may have dozens or hundreds of modules and
they can be categorized into different categories.

It is similar to the situation in a file system:

Instead of having all files in just one directory,
you put them into different ones,
being organized according to the topics of the files.

organize modules into packages.

https://python-course.eu/python-tutorial/modules-and-modular-programming.php