Programmieren I (Python)

Christian Osendorfer

2023-11-16

Modules

Modularization

- Organize your program by splitting up the code into various parts (so called *modules*)
- Modules are handled in Python with two statements
  - `import`: Import a module as a whole
  - `from ... import ...`: import single **names** (classes, functions) from a module.
- Python differentiates between to types of modules:
  - Libraries
  - local modules: only available for a local program.

Modularization

- Modules allow to organize code in a complex system and make components available through the respective **name space**.
- Names/Variables (that is, functions and classes) at the top-level of a module become attributes of the imported **module object**.
- Modules have at least 3 roles:
  - reuse of code.
  - partitioning of the namespace.
  - implementation of shared services and data.
- Modules are usually not executed directly.

Structure of a python program

- A Python program usually consists of multiple files, with one file playing the role of the *main file* (that's not an official name!).
- We start the program with the main file (`python my_main_file.py`), which usually has the main control logic of the program.
- Module files are libraries of tools (functions or classes) that are used by the main file or other modules.

Namespaces of modules

- Importing a module (a library) makes its attributes available through a new namespace

```
1 import math
2
3 math.exp(0)
```

- One can also directly import attributes, but this means loosing namespacing!

```
1 from math import pi # you can even do from math import * -> try to avoid it...
2 pi
```

- Rename namespaces for maximum flexibility

```
1 import math as m
2 m.exp(0)
```

dir

- `dir` is a function in the standard library of Python and shows all available names

```
1 import math
2 print(dir(math))
3 # what is this "__name__"?
4 print(math.__name__)
5 print(help(math)) # what is this doing?
```

What is `import` doing?

- `import` does not mean that the respective module is copied into a file as a text. The imported module is an object (everything is an object in Python).
- `import` is a runtime operation
  - find the module file
  - compile module to bytecode (if necessary)
  - execute the module code (that is, generate the various objects)

Searching for a module

- Python looks for the module
  - in the current folder
  - in the folders defined in `PYTHONPATH` (a shell/OS variable)
  - if `PYTHONPATH` is not defined, look through all folders in `sys.path`

```
1 import sys
2 for p in sys.path:
3   print(p)
```

The `__name__` attribute of a module

- When we **run** a module (that is, `python my_program.py`), the Python interpreter overrides the default module `__name__` and sets it to the special string `'__main__'`.
- Writing `if __name__ == '__main__':` in a module means:
  - Execute the following code if the module is being run …
  - and ignore the following code if the module is imported by another module.

Organizing a Python file

- We call the `if` branch under the `if __name__ == '__main__':` the **main block** of a module.
- Conventions:
  - The only code that goes outside of the main block of a module are import statements (`import ...`), constant definitions (`MY_CONSTANT = ...`), function definitions (`def ...`), and data type definitions (`class ...`)
  - Other code, like code for running doctest, reading command line parameters, etc goes inside the main block so that it is only executed when the module is run, and not when it is imported.
  - The main block goes at the bottom of the module.

Python Package

- A **package** collectes several, related modules together. It is used like a module! (`import package`).
- A package is simply a folder (that can be found by the Python interpreter …) which has a file `__init__.py`
  - `__init__.py` can be empty (!) or contains code which is executed when the package is imported.
  - Add modules to this folder as reasonable.
- Packages can have more sub-packages (that is, subfolders with a `__init__.py` file)

`__init__.py`

- `__init__.py` is executed when its package is `import`ed.
- Initializes the namespace of the module (paths in folder structures are translated to object paths)
- Can define a `__all__` variable which is a list of the package's objects which are exported when doing a `from package import *`.