COMP1730/COMP6730
Programming for Scientists

Modules and programs

Announcements

- ⋆ Major assignment due next Monday - 9:00am.
- ⋆ No new lab exercises this week.
- ⋆ Examination revision labs next week.

Lecture outline

- ★ python modules & import
- ★ Command-line interface and scripting
- ★ User interaction

Modules

Modules

- ⋆ Every python file is a module.
  - – A module is a sequence of statements.
  - – Every module has a name.
- ⋆ When the python shell runs in "script mode", the file it's executing becomes the "main module".
  - – Its name becomes $'$__main__$'$.
  - – Its namespace is the global namespace.
- ⋆ The first time a module is imported, that module is loaded (executed); it may later be re-loaded.
- ⋆ Every loaded module creates a separate (permanent) namespace.

* When executing import *modname*, the python interpreter:
  – checks if *modname* is already loaded;
  – if not (or if reloading), it:
    – finds the module file (normally *modname*.py)
    – executes the file in a new namespace;
    – and stores the module object (roughly, namespace) in the system dictionary of loaded modules;
  – and then associates *modname* with the module object in the current namespace.
* Note: the Spyder IDE reloads all user-defined modules on (first) import when running a file.

- ★ The global variable `__name__` in every module namespace stores the module name.
- ★ `sys.modules` is a dictionary of all loaded modules.
- ★ `dir(module)` returns a list of names defined in *module*'s namespace
- ★ `dir()` lists the current (global) namespace.

```
>>> __name__
'__main__'
>>> import sys
>>> len(sys.modules)
...
>>> sys.modules['math'].__name__
'math'
>>> dir()
[ ..., sys ]
>>> import math
>>> dir()
[ ..., sys, math ]
```

```
def some_useful_function(x):
    ...
if __name__ == '__main__':
    # this part will not execute when
    # the module is imported
    print(some_useful_function(0))
    ...
```

★ Code within the if statement will execute when the module is run, but not when it's imported ("guarded main").

★ For example, test cases.

The commandline

* A commandline ("terminal" or "shell") is a text I/O interface to the computer's operating system (OS).

* The shell is an *interpreter* for a command (programming) language.

(Image from wikipedia)

* The languages of shells are (more or less) different, but some aspects are fairly common.

* Some concepts from the commandline interface explain how programs interact with the OS.

- ★ To run a (executable) program, type its name.
  - – Where the OS searches for programs is usually configurable.
  - – Alternatively, enter the full path.

- ★ To run a python program (file):

  $ python3 *my_prog*.py

  - – Runs the python shell in "script mode".

- ★ Can pass arguments (strings) to the program:

  $ python3 *my_prog*.py *arg1* "arg two"

- ⋆ Inputs that the OS provides to the program:
  - **–** A list of commandline arguments (strings).
  - **–** A set of *environment variables* (key–value pairs, both (byte) strings).
  - **–** Open files (or file-like objects) for "standard input" and "standard output".
- ⋆ You can access these within python:
  - **–** `sys.argv`
  - **–** `os.environ` and `os.getenv(var)`
  - **–** `sys.stdin` and `sys.stdout`
- ⋆ By default, `input(..)` reads `sys.stdin` and `print(...)` writes to `sys.stdout`.

User interaction

* A general-purpose program (not solving a single instance of a single problem) will need some user input. For example:
  – which data file? computation parameters;
  – options (e.g., more or less output).
* Main goal: *don't make the user's life harder than it has to be.*
  – Know the use case; follow conventions.
  – Reduce work, avoid repetition.
  – Offer flexibility, but not at the cost of simplicity.
* If you're writing a library (module), the "user" is the programmer that will use its functions.

Example: Asking for a filename

- ★ Make it a commandline argument.
  - **–** Use `argparse` or `getopt` module for commandline processing.
- ★ Typed input (`input(..)`)
  - **–** Can use, and also customise, the `readline` facility, or use the `prompt_toolkit` module (system-dependent).
  - **–** Can provide defaults or shortcuts.
- ★ Open a "Select File" dialog box, using `tkinter` (system-dependent).

- ⋆ Example: the homework testing program.
- ⋆ Needs an input (the file to test).
  - – Fixed name (edit program to change).
  - – Typed input?
  - – Dialog box?
- ⋆ Who are the users, and what are their use cases?
  - – Student: testing one file, many times.
  - – Marker: testing many files, once.