ECE326

PROGRAMMING LANGUAGES

Lecture 5a : Python Modules

Kuei (Jack) Sun

ECE

University of Toronto

Fall 2020

Code Organization

- Module
  - A Python source file
    - Contains a collection of definitions and statements
  - Prevents name conflict
    - E.g. `math.abs` vs. `bodybuilder.abs`
    - Similar to C++ namespace
  - Allows for code reuse
    - Python is known for its comprehensive standard library
    - “Batteries included”

Import

- Gains access to definitions inside module
  - No information hiding, everything is accessible
- Use name of file (minus the .py)
  - E,g, to import foo.py in same folder, use `import foo`
  - File executed when importing
- Python standard library
  - https://docs.python.org/3/library/
  - E.g. `socket, struct, io`

Python Idiom

- Avoid execution if imported as module

foo.py

```
print(“hello”, __name__)
def unittest():
    print(“unit testing”)
# will not run if imported as module
if __name__ == “__main__”:
    unittest()
```

main.py (we run this file)

```
import foo                  # prints “hello foo”
print(type(foo))            # prints <type ‘module’>
print(foo.__name__)         # prints “foo”
print(__name__)             # prints “__main__”
```

Import

- A Python statement
- Can be called anywhere in code
  - Convention: prefer at top of source file
  - Optimization: avoid import until just before use

```
def unlikely_called_function():
    import huge_module
    huge_module.do_something()
```

- Python tracks which module already imported
  - Same module will not be re-imported

From

- Import name into local namespace
  - Don’t have to prefix with module name

```
import struct
from struct import Struct

# must say struct.pack instead of just pack
print(struct.pack("ii", 15, 32))

 # Struct is in local namespace, don't need prefix
packer = Struct("ii")
print(packer.pack(16, 31))
```

Output:

```
b'\x0f\x00\x00\x00\x04\x00\x00\x00'
b'\x10\x00\x00\x00\x1f\x00\x00\x00'
```

Best Practice

- import *
  - Imports everything from module into local namespace
    - Except “private” variables! (name starts with _)

```
from bar import *
```

  - Can become a source of bugs for long term projects
    1. Can overwrite existing name
    2. Can hide problems in code
- Compromise
  - `__all__`
    - Explicitly define what names are exported on import *

Package

- A directory can also be imported (a.k.a *package*)
  - Good way to organize multiple modules
  - Must have a special file named __init__.py
  - To import a file in *package*, must prefix with package name

```
import easydb.exception
```

  - Use the as operator to replace fully qualified name

```
import easydb.exception as ex
```

  - Can drop package name for from *xxx* import *yyy*

```
from .exception as IntegrityError
```