Structuring your Code

Shankar Kulumani

Flight Dynamics & Control Lab

THE GEORGE WASHINGTON UNIVERSITY

WASHINGTON, DC

Structure of Python Code

- Your Python code is made up of several components
  - Lines of variables/evaluations
  - Functions - several lines which achieve a single purpose
  - Modules - several related functions (file)
  - Packages - several related modules (directory)
- It helps to create a standard structure so your code is organized and easy to use and modify
- Also much of the automated testing and other features depend on a clear structure for you code

Basic Structure Example

- The `astro` library is a good example of the basic structure

```
README.md
driver.py
package_a
    __init__.py
    module_a.py
    module_b.py
package_b
    __init__.py
    module_a.py
tests
    __init__.py
    test_package_a.py
    test_package_b.py
```

Structure of Code is Key

- Avoid Circular dependencies - modules and packages should try to be independent
- Hidden coupling - making a small change breaks many other functions/tests
- Passing variables instead of globals - many objects can modify a global variable

Modules

- One of the main abstraction layers - separate code into parts holding related data and functionality
  - One layer might handle user input
  - Another layer handles low-level data manipulation
  - Group each into separate files
  - Python `import` and `from ... import` statements
- Module names should be short, lowercase, and avoid special symbols
  - Nothing else special is required - it's just a file with a fancy name now
  - Python will search for the file - and make it available using the `import` statement

Best Practices

Very Bad

```
[...]
from modu import *
[...]
x = sqrt(4) # where did sqrt come from?
```

Better

```
from modu import sqrt
[...]
x = sqrt(4) # sqrt might be part of modu, if not redefined
```

Best

```
import modu
[...]
x = modu.sqrt(4) # sqrt is visibly part of modu
```

Readability and Clarity are important

Best Practices

Very Bad

```
[...]
from modu import *
[...]
x = sqrt(4) # where did sqrt come from?
```

Better

```
from modu import sqrt
[...]
x = sqrt(4) # sqrt might be part of modu, if not redefined
```

Best

```
import modu
[...]
x = modu.sqrt(4) # sqrt is visibly part of modu
```

Readability and Clarity are important

Best Practices

Very Bad

```
[...]
from modu import *
[...]
x = sqrt(4) # where did sqrt come from?
```

Better

```
from modu import sqrt
[...]
x = sqrt(4) # sqrt might be part of modu, if not redefined
```

Best

```
import modu
[...]
x = modu.sqrt(4) # sqrt is visibly part of modu
```

Readability and Clarity are important

Best Practices

Very Bad

```
[...]
from modu import *
[...]
x = sqrt(4) # where did sqrt come from?
```

Better

```
from modu import sqrt
[...]
x = sqrt(4) # sqrt might be part of modu, if not redefined
```

Best

```
import modu
[...]
x = modu.sqrt(4) # sqrt is visibly part of modu
```

Readability and Clarity are important

Packages

- Packages are a simple extension to modules
- Just place an empty file named `__init__.py` in a directory
- Same import structure as modules

Convenient syntax for deeply nested packages

```
import very.deep.module as mod
```