Presentation Schedule

Monday May 4:
1:10: Breunig, Hoffland, Lammer, Noel
1:30: Ahola, Krouth, Oswald, Persin
1:50: Boeck, Gruber, Thoman

Wednesday May 6:
1:10: Detaege, Fredrickson, LaFleur
1:30: Lynaugh, Perrin, Veldboom
1:50: Fitzpatrick, Witthun, Writz

Friday May 8:
1:10: Watson, Xiong
1:30: Bedford, LaRue, Milos
1:50: Agarano, Hirt, Pierick
Consider these modules

- engine.py
- transmission.py
- electrical.py
- brakes.py
- unibody.py

Cars

Trucks

- engine.py
- transmission.py
- electrical.py
- brakes.py
- frame-body.py

Motorcycles

- engine.py
- transmission.py
- electrical.py
- brakes.py
- frame.py

We need to use all of these in a single project but the names collide.

Possible Solutions

- Use `import as` to rename the modules at runtime
  - This is rather ad hoc

- Rename the modules by changing filenames
  - Could cause problems with others using the modules

- Find work in another industry
  - It won’t pay as well as CS unless you go into business but the thought of doing that makes your blood run cold

- Manage the modules in a package
  - Please, Dr. Mathias, tell us more!
What is a package?

- It’s just a collection of modules
  - (see slides from Class 30)

- It can be hierarchical
  - i.e. contain subpackages
  - a subpackage is a package within another package

- Can contain (limited) runnable code outside of the modules

- It’s implemented as a directory on disk
  - package name is determined by directory name
  - (analogous to a module name being determined by .py filename)

Let’s package-ize our example

Top-level directory and package name

Subdirectories and subpackages

Vehicles/
  Cars/
    engine.py
    transmission.py
    electrical.py
    brakes.py
    unibody.py

Trucks/
    engine.py
    transmission.py
    electrical.py
    brakes.py
    frame-body.py

Motorcycles/
    engine.py
    transmission.py
    electrical.py
    brakes.py
    frame.py
Can nest more deeply as needed

Vehicles/

Cars/

Drivetrain/

   engine.py
   transmission.py

   electrical.py
   brakes.py
   unibody.py

Subpackage of subpackage Cars

Similar to functional decomposition, logical structure of your problem will determine the hierarchy of your packages

One more detail

A package can (should?) include an __init__.py file to execute initialization code:

Vehicles/

__init__.py

Cars/

__init__.py

Drivetrain/

   __init__.py
   engine.py
   transmission.py

   electrical.py
   brakes.py
   unibody.py

More on this in a moment
Using a package

- To use the package, import some or all of its modules in another program

```python
import Vehicles.Cars.brakes
Vehicles.Cars.brakes.absTest()

from Vehicles.Cars import brakes
brakes.absTest()

from Vehicles.Cars.brakes import absTest
absTest()
```

Using a package

- When you import any part of a package, its \_init\_.py is executed

```python
import Vehicles.Cars.brakes
Vehicles.Cars.brakes.absTest()

from Vehicles.Cars import brakes
brakes.absTest()

from Vehicles.Cars.brakes import absTest
absTest()
```

All of these cause: Vehicles/\_init\_.py and Vehicles/Cars/\_init\_.py

to execute
Importing an entire package

```python
import Vehicles

Vehicles.Cars.brakes.abs_test()  # ERROR
```

This doesn’t import anything unless...

__init__.py

- Because __init__.py is run when we load a module, we can use it to control imports

```python
# Vehicles/__init__.py
from . import Cars, Trucks, Motorcycles

# Vehicles/Trucks/__init__.py
from . import engine, transmission, electrical, brakes
```
### __init__.py

```python
# Vehicles/__init__.py
from . import Cars, Trucks, Motorcycles

# Vehicles/Trucks/__init__.py
from . import engine, transmission, electrical, brakes

import Vehicles       # now this call imports the modules
                      # enumerated in the init files
```

### Wildcard import

- Recall that in a module, we can define __all__ to control the elements that are imported using *
- Similarly, we can define __all__ in a package to control the modules that are imported using *

```python
# Vehicles/Trucks/__init__.py
__all__ = [engine, transmission, electrical, brakes]

# code that uses the package
from Vehicles.Trucks import *
```
Relative `import`

- Some modules in a package may need to import elements from other modules in that package

```python
# Vehicles/Trucks/transmission.py
from ..Vehicles.Cars import unibody
```

.. indicates up one level