I needed to write a lecture for today but...

...I was distracted by important events like this

I needed to write a lecture for today but...

...and this

Announcements

• Wednesday we will meet in lab
  • This is a change from what I announced last Friday

• Reminder: Friday is the next project milestone
  • I will meet with each group to get an update
    • Current state of the project
    • What work remains
    • Changes to project scope
    • Timeline for completion (due: May 5)
Quiz 5

class Car(object):
    num_cars = 0

    def __init__(self, make='Ferrari', model='458 Italia'):
        self.make = make
        self.model = model
        num_cars += 1

    def __del__(self):
        num_cars -= 1

    def print_car(self):
        print(' make: {}
          model: {}'.format(self.make, self.model))

class SportsCar(Car):
    def __init__(self, make='Ferrari', model='458 Italia', hp = 400):
        Car.__init__(self, make, model)
        self.hp = hp

    def print_car(self):
        Car.print_car(self)
        print(' hp: {}'.format(self.hp))

Consider these modules

cars
engine.py
transmission.py
electrical.py
brakes.py
unibody.py

trucks
engine.py
transmission.py
electrical.py
brakes.py
frame.py

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

- Vehicles/
  - Cars/
    - Drivetrain/
      - engine.py
      - transmission.py
      - electrical.py
      - brakes.py
      - unibody.py
    - Subpackage of subpackage Cars
  - Motorsports/
    - Engine:
      - engine.py
      - transmission.py
      - electrical.py
      - brakes.py
    - frame-body.py
    - Subdirectories and subpackages
  - Top-level directory and package name

Can nest more deeply as needed

- Vehicles/
  - Cars/
    - Drivetrain/
      - engine.py
      - transmission.py
      - electrical.py
      - brakes.py
      - unibody.py
    - Subpackage of subpackage Cars
  - 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()
```

**Vehicles/\_\_init\_.py**

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

```python
import Vehicles.Cars.brakes
Vehicles.Cars.brakes.abs_test()
from Vehicles.Cars import brakes
brakes.abs_test()
from Vehicles.Cars.brakes import abs_test
abs_test()
```

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

Using a package

**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
```
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