Package & Module Structure

Python Modules

Python’s module system is a delight - easy to use, well designed, and extremely flexible.

```python
from greputils import grepfile
grepfile("pattern to match", "/path/to/file.txt")
```

Let’s look at how it might evolve, from simple to rich and complex.

Start With A Little Script

```python
# findpattern.py
import sys

def grepfile(pattern, path):
    with open(path) as handle:
        for line in handle:
            if pattern in line:
                yield line.rstrip('\n')

pattern, path = sys.argv[1], sys.argv[2]
for line in grepfile(pattern, path):
    print(line)
```

This also creates a module called `findpattern`.

Reuse Some Code

```python
# finderrors.py
import sys
from findpattern import grepfile

path = sys.argv[1]
for line in grepfile('ERROR:', path):
    print(line)
```

```
$ python3 finderrors.py log1.txt
Traceback (most recent call last):
  File "finderrors.py", line 3, in <module>
    from findpattern import grepfile
  File "/Users/amax/wdir/courses/online-python-beyond-basics/inc/modules/findpattern.py", line
10, in <module>
    pattern, path = sys.argv[1], sys.argv[2]
IndexError: list index out of range
```

What’s the error?

Main Guard

The solution: use a "main guard".

Original tail of `findpattern.py`:

```
pattern, path = sys.argv[1], sys.argv[2]
for line in grepfile(pattern, path):
    print(line)
```

Replace with:

```
if __name__ == "__main__":
    pattern, path = sys.argv[1], sys.argv[2]
    for line in grepfile(pattern, path):
        print(line)
```

Now it works:

```
$ python3 finderrors.py log1.txt
ERROR: out of milk
ERROR: alien spacecraft crashed
```

Magic __name__

`__name__` is a magic variable set to "`__main__`" if it's in the main executable file, or the current module name otherwise.

For example:

```
# say_hello.py
print("__name__ in say_hello.py: " + __name__)
def greet(): print("Hello!")
if __name__ == "__main__":
    greet()
```

```
# use_say_hello.py
print("__name__ in use_say_hello.py: " + __name__)
from say_hello import greet
if __name__ == "__main__":
    greet()
```

What is printed out if you run each?

Magic __name__

```python
# say_hello.py
print("__name__ in say_hello.py: " + __name__)
def greet(): print("Hello!")
if __name__ == "__main__":
    greet()
```

```
$ python3 say_hello.py
__name__ in say_hello.py: __main__
Hello!
```

```python
# use_say_hello.py
print("__name__ in use_say_hello.py: " + __name__)
from say_hello import greet
if __name__ == "__main__":
    greet()
```

```
python3 use_say_hello.py
__name__ in use_say_hello.py: __main__
__name__ in say_hello.py: say_hello
Hello!
```

Separate Libraries

Let's refactor to have a common library, so we can add extra functions.

```python
# greputils.py
# Search for matching lines in file.
def grepfile(pattern, path):
    with open(path) as handle:
        for line in handle:
            if pattern in line:
                yield line.rstrip('\n')
# Case-insensitive search.
def grepfilei(pattern, path):
    pattern = pattern.lower()
    with open(path) as handle:
        for line in handle:
            if pattern in line.lower():
                yield line.rstrip('\n')
```

Then `findpattern.py` and `finderrors.py` will have the line:

```python
from greputils import grepfile
```

Expanding Libraries

Suppose `greputils` keeps adding functions, like `contains`:

```python
def contains(pattern, path):
    with open(path) as handle:
        for line in handle:
            if pattern in line:
                return True
    return False
```

(And also `containsi`, for case-insensitive matching.)

As we add more, at some point we'll want to split up `greputils.py`. How?

Multifile Modules

There's more than one way to provide `greputils`. Let's split it into multiple files:

```
greputils/
greputils/__init__.py
greputils/files.py
greputils/contain.py
```

The `grepfile` and `grepfilei` functions are in `greputils/files.py`; `greputils/contain.py` has the `contains` and `containsi` functions.

The module directory generally must have an `__init__.py` file. This defines the interface for others importing the module.

__init__.py

```python
from .files import (
    grepfile,
    grepfilei,
)
from .contain import (
    contains,
    containsi,
    )
```

Note:

- Split over multiple lines, using parenthesis.
- Uses "`from .files import`". "`from grepfile.files import`" will also work, but is less maintainable.
- "`from files import`" works in Python 2 only. But it's ambiguous, which is why Python 3 doesn't allow it.

Nesting

You can break up into different folders however you like:

```
greputils/
greputils/__init__.py
greputils/files.py
greputils/contain.py
greputils/net/__init__.py
greputils/net/html.py
greputils/net/text.py
greputils/net/json.py
```

```python
# in greputils/__init__.py
# ...
from .net.html import (
   grep_html,
   grep_html_as_text,
)
```

Note the module interface doesn’t change!

Antipattern Warning!

Sometimes you will see this:

```python
from .files import *
from .contain import *
```

Don’t do that - ESPECIALLY in your application code. It lets collisions and subtle bugs sneak in.

More on __init__

`__init__.py` can, when it makes sense, execute init code.

In general, avoid import-time side effects, unless you have a good reason to.

`__init__.py` can be an empty file. In that case, users will import sub-modules:

```
from greputils import files
# Or:
import greputils.files
```

Importing

In your code you have a choice.

```python
from greputils import grepfile
grepfile("pattern to match", "/path/to/file.txt")
```

Versus:

```python
import greputils
greputils.grepfile("pattern to match", "/path/to/file.txt")
```

More Options

```
import greputils.contain

import greputils.contain as grepcontain

from greputils import files

from greputils.files import grepfilei as ci_grep

from greputils import (
    grepfile,
    contains,
    )
```

Terminology

The official terminology:

Reusable code in a single file is a **module**.

If that exact same code is split into multiple files, it’s called a **package**.

Lab: Create A Package

Lab file: `modules/modules.py`

- In labs/py3 for 3.x; labs/py2 for 2.7
- When you are done, give a thumbs up

See also: `modules/greputils_start.py`

Unlike other labs, you do NOT modify `modules.py` at all. Instead, create a `greputils` directory, and populate it as a package, using the functions in `greputils_start.py`.