Python Module Organization

Python Modules

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

```
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

```
# 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

```
# 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 "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 both programs work:

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

Separate Libraries

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

```
# 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:

```
from greputils import grepfile
```

Expanding Libraries

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

```
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

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

Note:

- Uses "`from .files import`", etc.
- Imported components split over multiple lines, using parentheses.
- "`from files import`" worked in ancient versions of Python. But it was ambiguous and a bug factory, so modern Python 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,
)
from .net.json import (
    grep_json,
    grep_json_many,
)
```

Another Choice

Another way to do it:

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

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

Antipattern Warning!

Sometimes you will see this:

```
from greputils import *
from otherlib import *
```

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

Importing

In your code you have a choice.

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

Versus:

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

Use this to namespace when you're using many objects from the module.

Renaming Imports

When importing the module to namespace, use an abbreviation to save yourself some typing.

```
# Commonly used:
import numpy as np
import pandas as pd
import tensorflow as tf
```

Also works with imported functions, classes, etc.:

```
# Import a function and rename it:
from greputils import grepfilei as ci_grep

# Similar to:
from greputils import grepfilei
ci_grep = grepfilei
```

More on `__init__.py`

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

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

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

```
from greputils.files import grepfile

# Or:
import greputils.files
for line in greputils.files.grepfile(pattern, path):
    # ...
```

In that situation, you can omit the `__init__.py` file completely! (This kind of module is called a "namespace package".)

Terminology

The official terminology:

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

A module with submodules is called a **package**. (So a package will be implemented as a directory, not a single file.)

From the perspective of the user, this is sometimes an implementation detail.

Lab: Create A Package

Lab file: `modules/modules.py`

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`.

IMPORTANT: Make sure your current directory is the one containing `modules.py` and your `greputils` folder. If you're using an IDE, it might be easier to run this lab on the command line.