Multi-File Python Programs

SURP 2022 Python Bootcamp

Ohio State Astronomy

Slides by: James W. Johnson

Import Statements: The Basics

Used to load python objects from a separate file or set of files

Analogous to *#include* and *using* in C/C++

Consider a simple example file to import:

```
r"""
Run "import simple_import" and also run this file to see what happens
differently between running and importing a python file.
"""

import math as m
import numbers


def f(x):
    r"""
    Calculate the value of x-squared
    """
    if isinstance(x, numbers.Number):
        return x**2
    else:
        raise TypeError("Must be a numerical value.")


print("The value of pi squared is %.5f" % (f(m.pi)))


if __name__ == "__main__":
    print("The value of e squared is %.5f" % (f(m.e)))

```

Import Statements: The Basics

Consider a simple example file to import:

```
r"""
Run "import simple_import" and also run this file to see what happens
differently between running and importing a python file.
"""

import math as m
import numbers


def f(x):
    r"""
    Calculate the value of x-squared
    """
    if isinstance(x, numbers.Number):
        return x**2
    else:
        raise TypeError("Must be a numerical value.")


print("The value of pi squared is %.5f" % (f(m.pi)))


if __name__ == "__main__":
    print("The value of e squared is %.5f" % (f(m.e)))

```

This is what happens when we run it:

```
(base) Cosmo:examples astrobeard$ python simple_import.py
The value of pi squared is 9.86960
The value of e squared is 7.38906
```

Import Statements: The Basics

Consider a simple example file to import:

This is what happens when we import it:

```
In [1]: import simple_import
The value of pi squared is 9.86960
```

if __name__ == "__main__" won't run if the file is imported

```
r"""
Run "import simple_import" and also run this file to see what happens
differently between running and importing a python file.
"""

import math as m
import numbers


def f(x):
    r"""
    Calculate the value of x-squared
    """
    if isinstance(x, numbers.Number):
        return x**2
    else:
        raise TypeError("Must be a numerical value.")


print("The value of pi squared is %.5f" % (f(m.pi)))


if __name__ == "__main__":
    print("The value of e squared is %.5f" % (f(m.e)))

```

Import Statements: The Basics

Other import formats:

```
In [1]: from simple_import import f
The value of pi squared is 9.86960

In [2]: from simple_import import m

In [3]: from simple_import import numbers
```

The file runs only once

```
"""
Run "import simple_import" and also run this file to see what happens
differently between running and importing a python file.
"""

import math as m
import numbers


def f(x):
    """
    Calculate the value of x-squared
    """
    if isinstance(x, numbers.Number):
        return x**2
    else:
        raise TypeError("Must be a numerical value.")


print("The value of pi squared is %.5f" % (f(m.pi)))


if __name__ == "__main__":
    print("The value of e squared is %.5f" % (f(m.e)))

```

The __init__.py file

Allows you to import an entire directory

This is how you create python packages

There is no file in your computer named numpy.py, but there is a file named numpy/__init__.py. The same goes for scipy, matplotlib, pandas, etc.

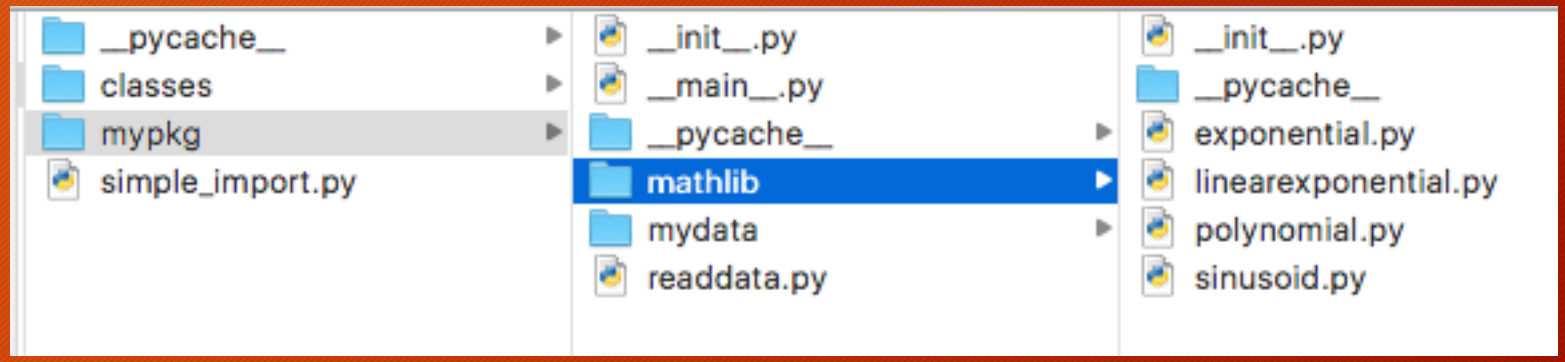
An Example Package

Located in the git repository at examples/mypkg

The __init__.py file:

```
__all__ = ["readdata"]
from .readdata import readdata
from .mathlib import *
__all__.extend(mathlib.__all__)
```

What this looks like in a Finder window:



An Example Package

The mathlib/__init__.py file:

```
__all__ = ["polynomial", "sinusoid", "exponential", "linearexponential"]
from .polynomial import polynomial
from .sinusoid import sinusoid
from .exponential import exponential
from .linearexponential import linearexponential
```

The mydata/__init__.py file:

```
1
2
3
4
```

Relative Imports

Special syntax telling python exactly where to look within your package

from .somefile import something
from ..somedirectory import something_else

The number of dots specifies the directory: one for same directory, two for parent directory, three for parent directory's parent directory, etc.

Blank __init__.py files are often included to note that the contents of a directory are still an important part of the package. These directories often contain, e.g., data files.

Other Contents of the __init__.py File

__all__: The names of the objects to import when you run *from ___ import **

Note: *from ____ import* * is considered bad practice anywhere other than inside of an __init__.py file. This is because it is possible to override function names and makes it difficult to keep track of your namespace.

Example: If I call *log10*, do I get *math.log10* or *numpy.log10*? What if I swap the order?

```
In [1]: from math import *

In [2]: from numpy import *
```

An Example Package

mypkg/__init__.py:

```
__all__ = ["readdata"]
from .readdata import readdata
from .mathlib import *
__all__.extend(mathlib.__all__)
```

mypkg/mathlib/__init__.py:

```
__all__ = ["polynomial", "sinusoid", "exponential", "exposinusoid",
    "linearexponential"]
from .polynomial import polynomial
from .sinusoid import sinusoid
from .exponential import exponential
from .exposinusoid import exposinusoid
from .linearexponential import linearexponential
```

mypkg/mydata/__init__.py:

```
1
2
3
4
```

mypkg.__all__ = [“readdata”, “polynomial”, “sinusoid”, “exponential”, …]

To Save You Time

If you are spreading code out across multiple files, and get an ImportError stating that a given package or module cannot be imported, you may have two files both importing one another in the preamble.

To solve this, move the import statement in one of the files to the functions which require it – there is no rule that says import statements need to be at the top; this is just convention.

The __main__.py File

The file that gets ran when you *run* a directory (i.e. *python mypkg*)

mypkg/__main__.py:

```
1
2  print("This is how you run a directory full of python code!")
3
4
```

```
(base) Cosmo:examples astrobeard$ python mypkg
This is how you run a directory full of python code!
```

Important: You *cannot* relative import in a *__main__.py* file

The Usefulness of your PYTHONPATH

Put useful code in a given directory or set of directories, put those directories on your PYTHONPATH, and import that code from *anywhere* in your computer.

You never have to write that code again! Take a guess how a software engineer would describe repeatedly writing similar code, even taking just a few seconds to copy and paste

The Usefulness of your PYTHONPATH

Put useful code in a given directory or set of directories, put those directories on your PYTHONPATH, and import that code from *anywhere* in your computer.

You never have to write that code again! Take a guess how a software engineer would describe repeatedly writing similar code, even taking just a few seconds to copy and paste

A waste of time