A Short Digression Into Multi-File Modules

`chardet` is a *multi-file module*. I could have chosen to put all the code in one file (named `chardet.py`), but I didn't. Instead, I made a directory (named `chardet`), then I made an `__init__.py` file in that directory. *If Python sees an `__init__.py` file in a directory, it assumes that all of the files in that directory are part of the same module*. The module's name is the name of the directory. Files within the directory can reference other files within the same directory, or even within subdirectories. (More on that in a minute.) But the entire collection of files is presented to other Python code as a single module — as if all the functions and classes were in a single .py file.

What goes in the `__init__.py` file? Nothing. Everything. Something in between. The `__init__.py` file doesn't need to define anything; it can literally be an empty file. Or you can use it to define your main entry point functions. Or you put all your functions in it. Or all but one.

> *A directory with an `__init__.py` file is always treated as a multi-file module. Without an `__init__.py` file, a directory is just a directory of unrelated `.py` files.*

Let's see how that works in practice.

```
import chardet
print (dir(chardet))                #①
#['__builtins__', '__cached__', '__doc__',
# '__file__', '__loader__', '__name__',
# '__package__', '__path__', '__spec__',
# '__version__', 'detect']

print (chardet)                     #②
#<module 'chardet' from '/usr/lib/python3/dist-packages/chardet/__init__.py'>
```

① Other than the usual class attributes, the only thing in the `chardet` module is a `detect()` function.

② Here's your first clue that the chardet module is more than just a file: the "module" is listed as the `__init__.py` file within the `chardet/` directory.

Let's take a peek in that `__init__.py` file.

```
def detect(aBuf):                              #①
    from . import universaldetector            #②
    u = universaldetector.UniversalDetector()
    u.reset()
    u.feed(aBuf)
    u.close()
    return u.result
```

① The `__init__.py` file defines the `detect()` function, which is the main entry point into the chardet library.

② But the `detect()` function hardly has any code! In fact, all it really does is import the `universaldetector` module and start using it. But where is `universaldetector` defined?

The answer lies in that odd-looking `import` statement:

```
from . import universaldetector
```

Translated into English, that means "import the `universaldetector` module; that's in the same directory I am," where "I" is the `chardet/__init__.py` file. This is called a *relative import*. It's a way for the files within a multi-file module to reference each other, without worrying about naming conflicts with other modules you may have installed in your import search path. This `import` statement will *only* look for the `universaldetector` module within the `chardet/` directory itself.

These two concepts — `__init__.py` and relative imports — mean that you can break up your module into as many pieces as you like. The `chardet` module comprises 36 `.py` files — 36! Yet all you need to do to start using it is `import chardet`, then you can call the main `chardet.detect()` function. Unbeknownst to your code, the `detect()` function is actually defined in the `chardet/__init__.py` file. Also unbeknownst to you, the `detect()` function uses a relative import to reference a class defined in `chardet/universaldetector.py`, which in turn uses relative imports on five other files, all contained in the `chardet/` directory.

> *If you ever find yourself writing a large library in Python (or more likely, when you realize that your small library has grown into a large one), take the time to refactor it into a multi-file module. It's one of the many things Python is good at, so take advantage of it.*