

Modules and Packages

There's no code here because it didn't really make sense for the section. Check out the video lectures for more info and the resources for this.

Here is the best source the official docs!

<https://docs.python.org/2/tutorial/modules.html#packages>
(<https://docs.python.org/2/tutorial/modules.html#packages>)

But I really like the info here: <https://python4astronomers.github.io/installation/packages.html>
(<https://python4astronomers.github.io/installation/packages.html>)

Here's some extra info to help:

Modules in Python are simply Python files with the .py extension, which implement a set of functions. Modules are imported from other modules using the import command.

To import a module, we use the import command. Check out the full list of built-in modules in the Python standard library here.

The first time a module is loaded into a running Python script, it is initialized by executing the code in the module once. If another module in your code imports the same module again, it will not be loaded twice but once only - so local variables inside the module act as a "singleton" - they are initialized only once.

If we want to import module math, we simply import the module:

```
In [5]: # import the library  
import math
```

```
In [6]: # use it (ceiling rounding)  
math.ceil(2.4)
```

```
Out[6]: 3
```

Exploring built-in modules

Two very important functions come in handy when exploring modules in Python - the dir and help functions.

We can look for which functions are implemented in each module by using the dir function:

```
In [8]: print(dir(math))
```

```
['__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'trunc']
```

When we find the function in the module we want to use, we can read about it more using the **help function**, inside the Python interpreter:

```
In [10]: help(math.ceil)
```

Help on built-in function ceil in module math:

```
ceil(...)
    ceil(x)
```

Return the ceiling of x as an int.
This is the smallest integral value $\geq x$.

Writing modules

Writing Python modules is very simple. To create a module of your own, simply create a new .py file with the module name, and then import it using the Python file name (without the .py extension) using the import command.

Writing packages

Packages are name-spaces which contain multiple packages and modules themselves. They are simply directories, but with a twist.

Each package in Python is a directory which MUST contain a special file called **__init__.py**. This file can be empty, and it indicates that the directory it contains is a Python package, so it can be imported the same way a module can be imported.


If we create a directory called foo, which marks the package name, we can then create a module inside that package called bar. We also must not forget to add the **__init__.py** file inside the foo directory.

To use the module bar, we can import it in two ways:

```
In [ ]: # Just an example, this won't work
import foo.bar
```

```
In [ ]: # OR could do it this way
        from foo import bar
```

In the first method, we must use the foo prefix whenever we access the module bar. In the second method, we don't, because we import the module to our module's name-space.

The `__init__.py` file can also decide which modules the package exports as the API, while keeping other modules internal, by overriding the `__all__` variable, like so: 

```
In [ ]: __init__.py:
        __all__ = ["bar"]
```