Project Structure and Imports

Project Structure and Imports

Overview

1. What Happens with `import`?
2. The Import Path
3. Structuring a Project

What Happens with `import`

What Happens with `import`

- `import` executes a module as if it were run from the command line
  - __name__ is the name of the module importing the module
  - Remember a module is simply a .py file
- Importing a file complies the file to bytecode in a .pyc file

What Happens with `import`

- `import` searches through the python path to find a module that matches the name specified
- Try running python from the command line with `-v` option and importing something
- But what is the path...

The Import Path

The Import Path

- The path that python searches is defined by `sys.path`
  - Not sure what yours is? Print it
- Imports will search in each location here, as well as in current directory
- These locations are defined by the PYTHONPATH system variable
- Add a location without changing the PYTHONPATH with a .pth file
- Add a temporary location by appending it to `sys.path`

Structuring a Project

Structuring a Project

- "Simplest" option is to keep everything together in one file
- Advantages:
  - Only have to keep track of one file
  - Do not need to worry about circular imports and other fun issues that arise when things are not in one file

Structuring a Project

- We can take advantage of imports and break a project into multiple files
  - A different file for each class
  - A different file for each type of class
  - A different file for functions grouped by type
  - A file for the interface
  - etc.

Structuring a Project

- Such an approach makes a project easier to maintain, more organized, and more self-documenting
    - e.g., files have names
- Downsides:
    - Losing a files breaks everything
    - Circular imports and other fun issues that arise when things are not in one file

Structuring a Project

- If you want the best of both worlds, create packages from small, like-minded pieces of your codebase
    - Increased modularity means small components can be broken out and made into packages
    - Then only need to maintain a single package instead of code copied across many projects
- Upload your packages to pypi and create requirement.txt files for your scripts
- pip will install your packages from pypi to the user's python installation, they will be in a location on the path, and your script can simply import them like any other package
- Perhaps a downside: you code will be publicly available.
    - In an enterprise situation with propriatry code, you can put your python package on a server, and have that path in your requirements.txt file rather than just the name.