**Modules and Packages**

**Definition**

Module -> Simply a Python file that holds related functionality. For example, if you have an e-commerce website you are likely doing a lot of DB queries. Those DB queries might be saved in a module named database.py

Package -> A collection of modules in a folder. Name of the package is the name of the folder. To create this in Python, the folder must have the \_\_init\_\_.py file.

**Relative vs. Absolute Imports**

Absolute imports specify the concrete path to the module, function or class to import. Assuming the below file structure

parent\_directory/

main.py AND ecommerce/

\_\_init\_\_.py database.py AND products.py AND payments/

\_\_init\_\_.py square.py stripe.py

1. Absolute Imports

Specify the complete path to the module, function or class to import. If we need to access Product class inside product module, we use the following. We assume that we are starting from parent\_directory

Import ecommerce.products

Product = ecommerce.products.Product()

OR

from ecommerce.products import Product

Product = Product()

OR

From ecommerce import products

Product = products.Product()

1. Relative Imports

It seems redundant to specify full path when working with related modules inside a package. This is where relative imports come in. Relative imports basically is a way to find a class, function or module as it is positioned relative to the current module.

If we are in the products module [which is in the ecommerce package], and we want to import the Database class from the database module, we use the following relative import

From .database import Database -> .database means from the same package, in a file called database, take class or function called Database.

If we were in the payments package we would want to use the database package inside the parent package this is done with this

From ..database import Database -> 2 periods means we want to go up 1 package.

**Bad Practices**

It is always better to explicitly import things don’t use syntax like this

From database import \*

Assuming a module name database exists in the same path. You should never use the above

**Organizing module content**

Do the below for a way to store global state without namespace conflicts. For example, instead of importing Database class into various modules and instantiating them, better to have 1 database object globally availbale from database module. Implement like this

In database.py

class Database:

# the database implementation

pass

database = Database()

Then in other packages

From ecommerce.database import database.

All module code is executed immediately at the time it is imported. However, if inside a method or function, the function will be created but internal code not exceuted till function is called.

The problem with the above implementation is that database object is created immediately when module is first imported which is when program starts up. This isn’t ideal as connecting to database takes a while and it slows down the startrup or the database connection finromation may not be availbe yet. We can delay creating the database till it is actually needed by doing the below:

class Database:

# the database implementation

pass

database = None

def initialize\_database():

global database

database = Database()

Then we call function initialize\_database() in other modules whenever it is needed.

**Main.py best practices**

The fact that