**Module and Packages**

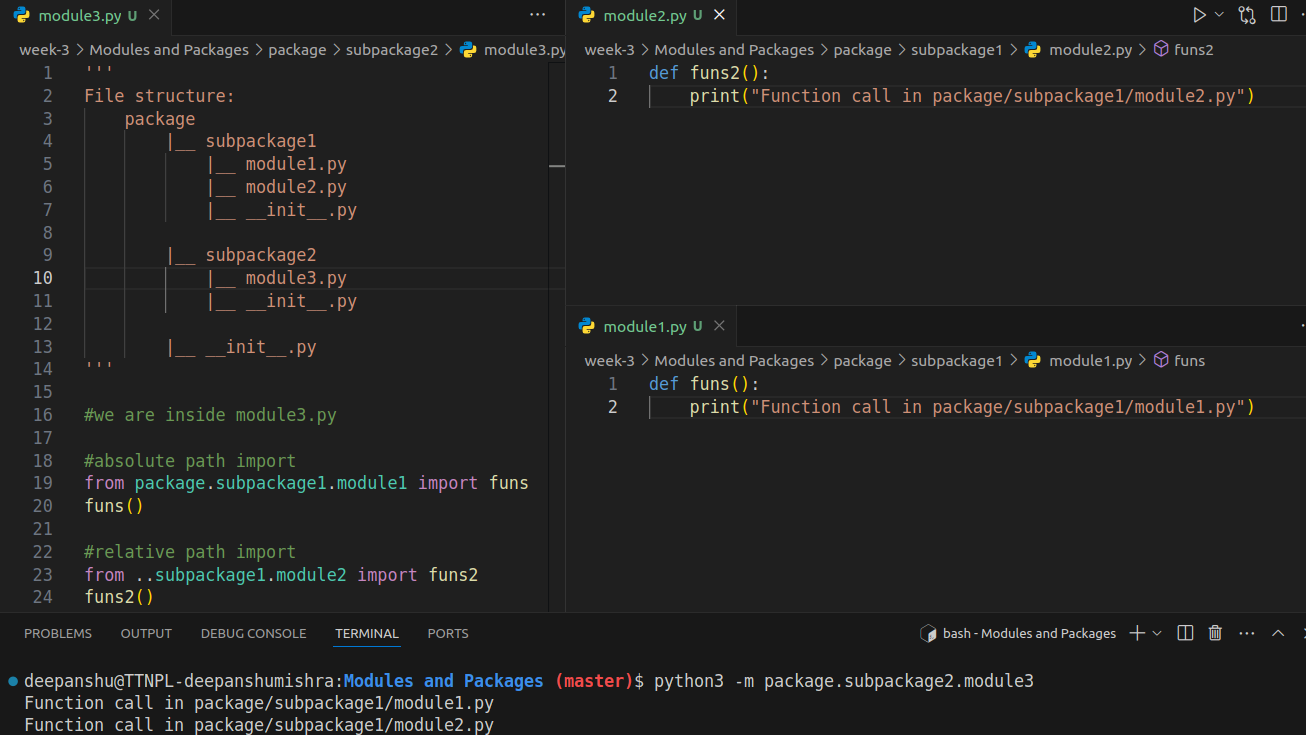
https://docs.google.com/document/d/1eBwGEZ1hwOjVFLTlPckdAL36uEpYjMGbiPSWKtlmt8I/

Q1)Write Python code scripts to demostract absolute vs relative imports.

In the following module3.py file funs() is called using the absolute path package.subpackage1.module1

And funs2() is called using a relative path. This path is declared using “.” to navigate upto a parent package and then going down to module2 where .. means that we need to look for subpackage1 inside the parent of the current directory

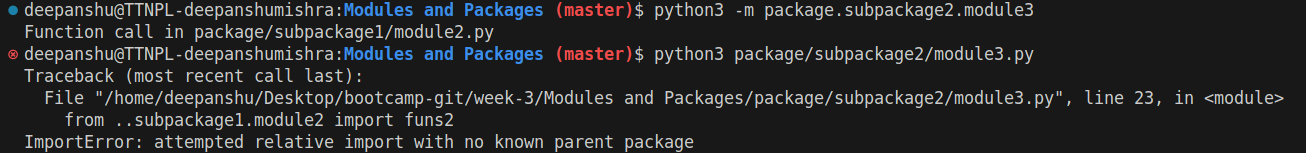
..subpackage1.module2



Prrof that relative imports should be explicit.

This statement likely refers to the concept of calling package modules explicitly that means the module(or a python script) cannot be executed itself if it contains a relative path, this is to provide programmers with ease of writing code within a package without having to worry about default script behaviours. Due to this, the module can only be called from outside a built package (using python3 -m from terminal)

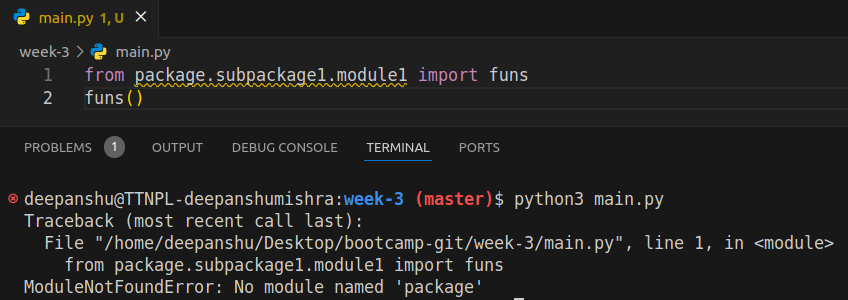
Let’s see what happens when we try to run module3 which contains a relative path import:



We encounter an error!

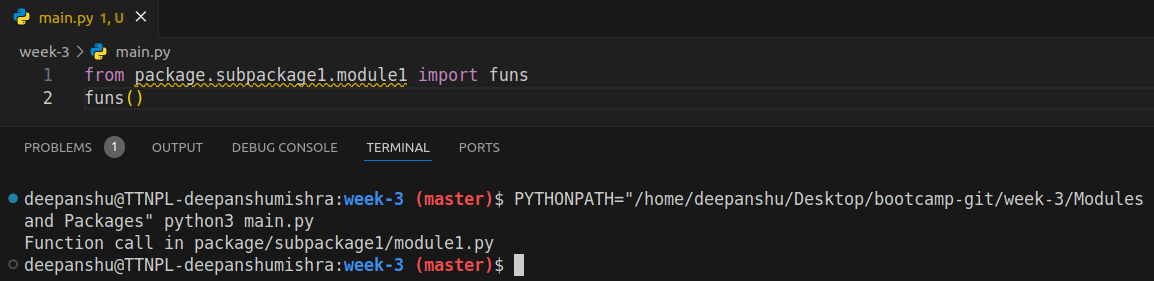
Also demostrat use of PYTHONPATH

PYTHONPATH is an environment variable in python that specifies directories where the interpreter should look for packages and modules. We use it when we have custom modules and packages that are not a part of the python standard library. By default to search for packages some directories are already configured in sys.path which also contains a pwd configuration which means even if we create a package in a project, we can still access it from within the project root directory. Lets create a main.py outside of the root directory and import a package module, lets see what happens.



Python cant find the package. Now we can use PYTHONPATH variable to set project’s root directory to look for packages. We can do it directly through shell(as demonstrated below) or we can add the same line export in ~/.bashrc file to permanently add it to PYTHONPATH.

Lets add it for a current session for main.py file and lets see if it gets executed

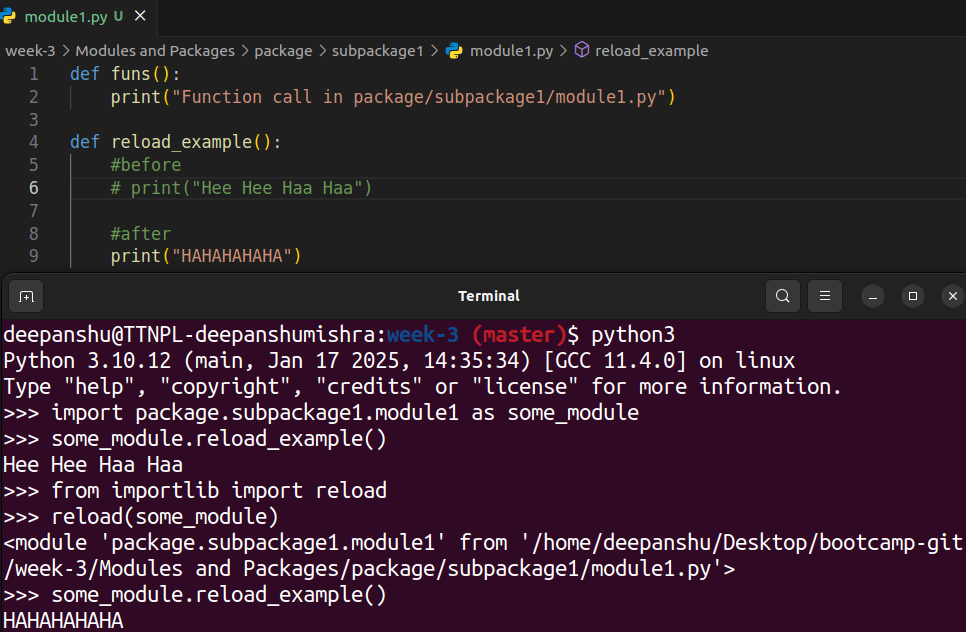


*Et Voila!*

Explain the use of `from importlib import reload`

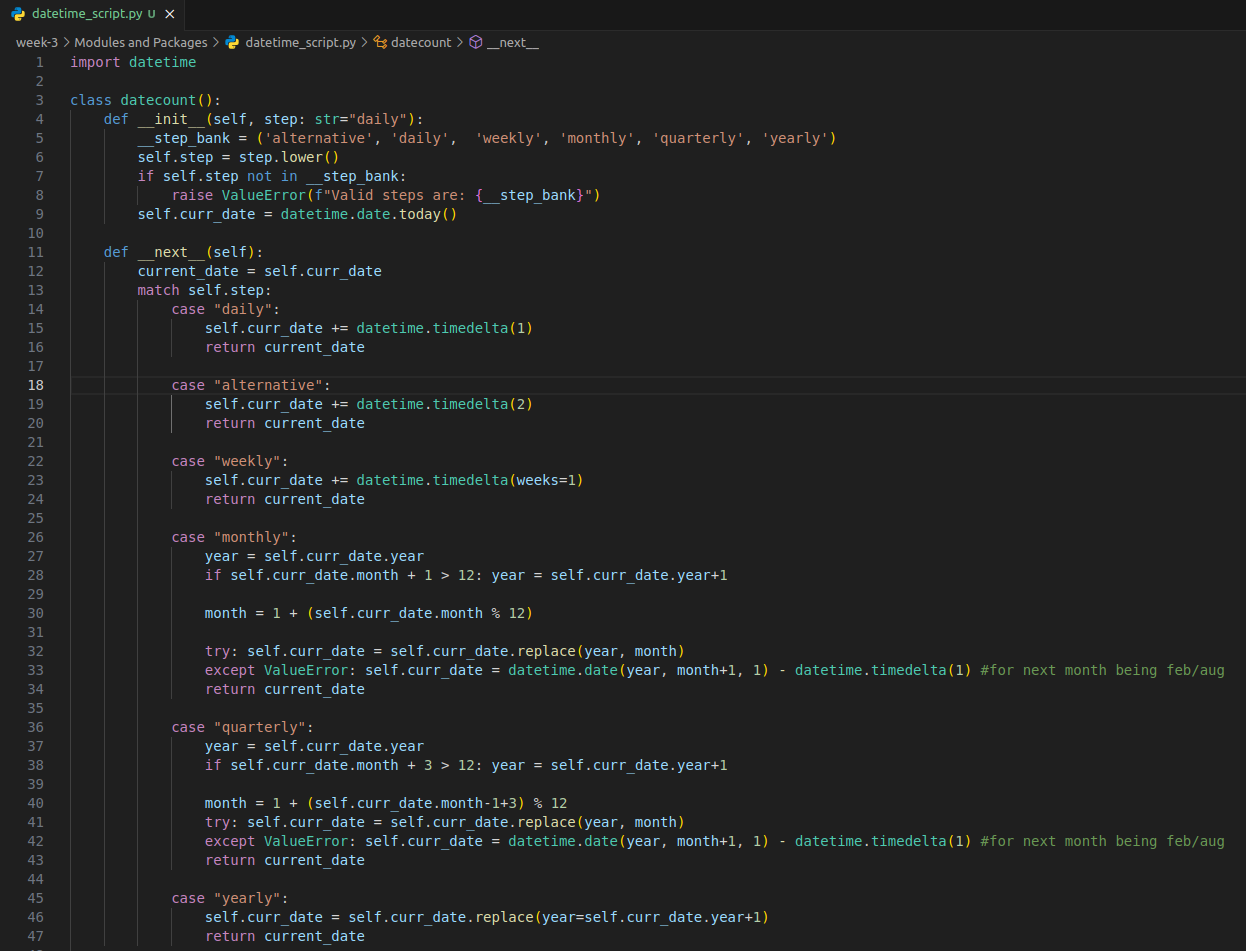
reload function from the importlib library is used often when we need to apply new changes to a module in the current running python session without restarting the session.

To understand this lets add a reload\_example function to a previous module that prints some text and run that in the interpreter. Now change the print text. Now without reloading we still see the old text, so now we import reload() function, pass the module as the argument in that function. We see a success text, now run the example again and now we see the new text!

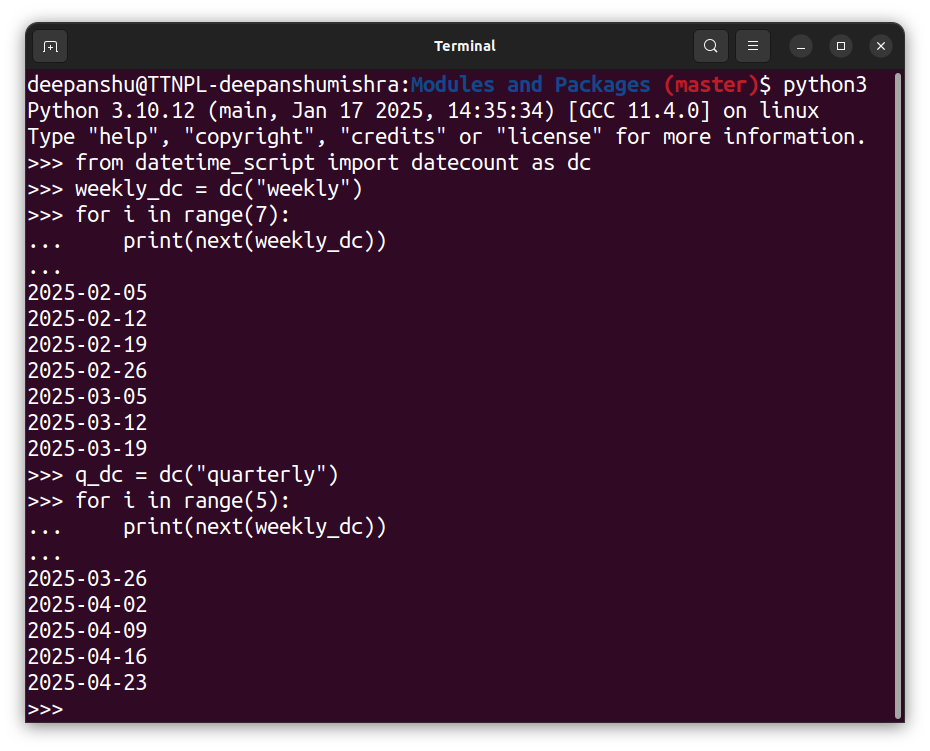


Read about itertools.count(start=0, step=1) function which accepts options arguments start and end Based on this, implement a similar `datecount([start, step])` where start is a `datetime.date` object and step can we string values 'alternative', 'daily', 'weekly', 'monthly', 'Quarterly', 'yearly' (ignore case)

Implementation using class



Usage of this datecount class in the interpreter. (python classes should be caps but followed question format)



Implementation using generator function and yield method

