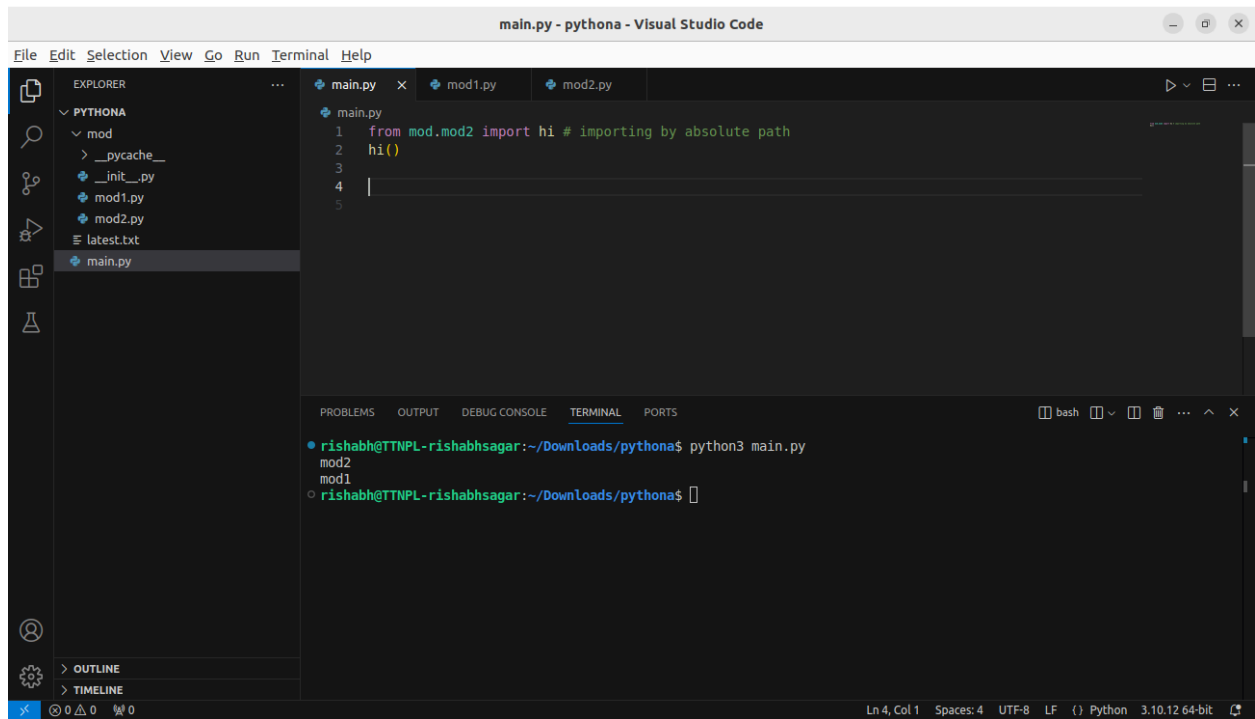


Module and Packages

Q1)Write Python code scripts to demonstrate absolute vs relative imports. Prrof that relative imports should be explicit. Also demonstrate use of PYTHONPATH
Ans.

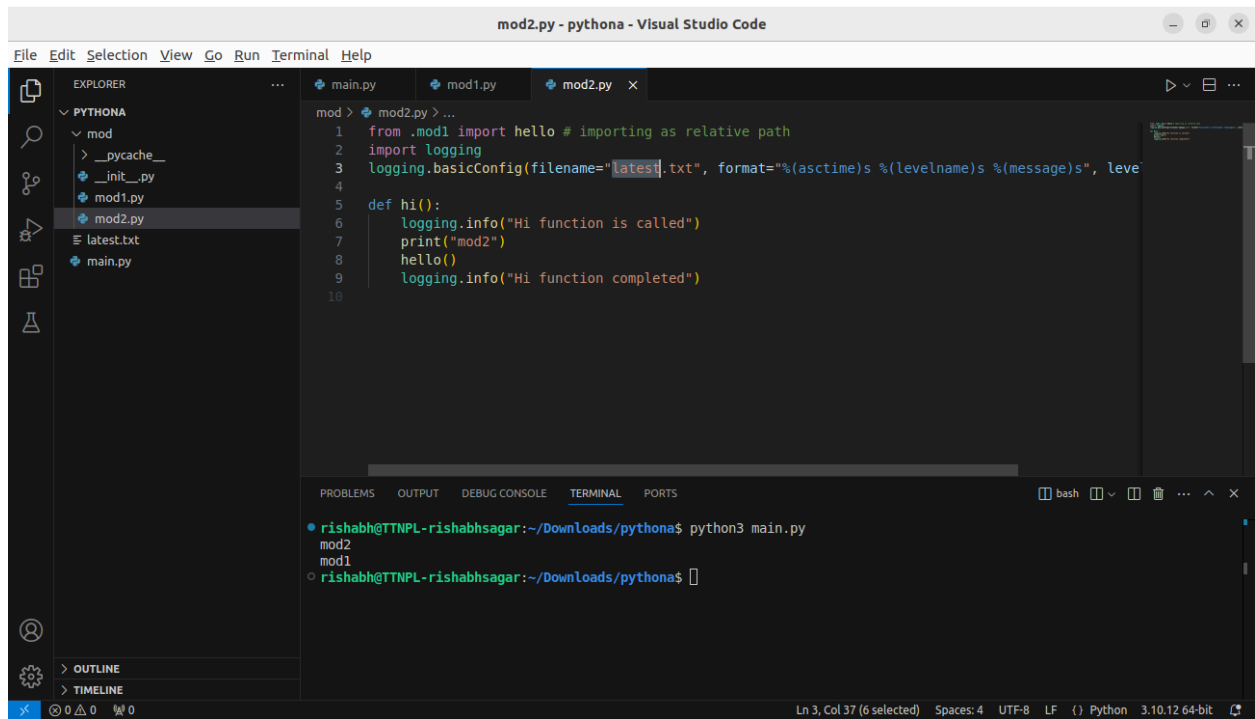


The screenshot shows the Visual Studio Code interface with a file explorer on the left containing a 'PYTHONA' package with sub-packages 'mod' and 'mod2'. The 'main.py' file is open in the editor, showing the following code:

```
1 from mod.mod2 import hi # importing by absolute path
2 hi()
3
4
5
```

The terminal at the bottom shows the command `python3 main.py` being executed, resulting in the output:

```
mod2
mod1
rishabh@TNPL-rishabsagar:~/Downloads/pythona$
```

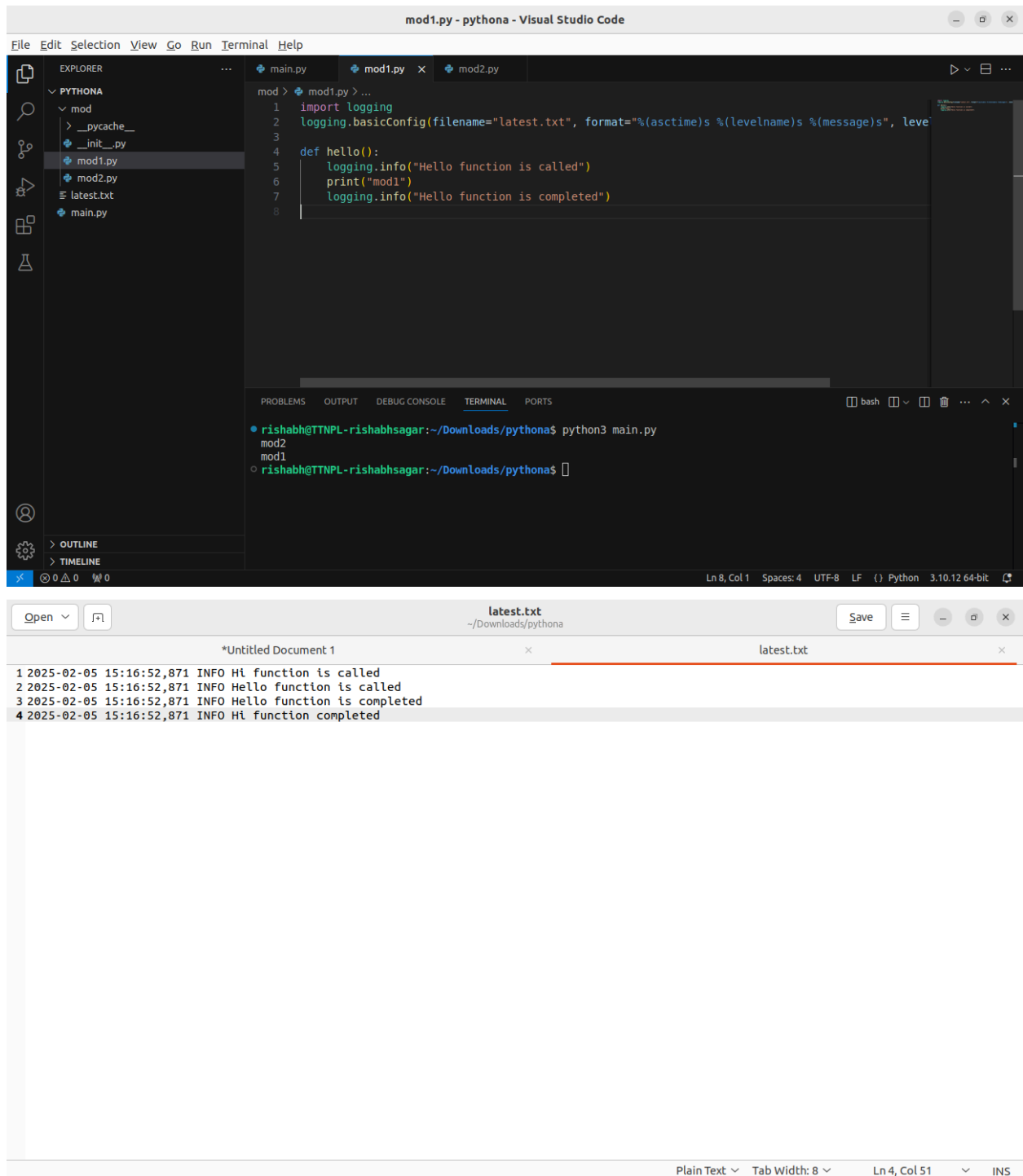


The screenshot shows the Visual Studio Code interface with the 'mod2.py' file open in the editor. The code is as follows:

```
1 from .mod1 import hello # importing as relative path
2 import logging
3 logging.basicConfig(filename="latest.txt", format="%asctime)s %(levelname)s %(message)s", level=logging.INFO)
4
5 def hi():
6     logging.info("Hi function is called")
7     print("mod2")
8     hello()
9     logging.info("Hi function completed")
10
```

The terminal at the bottom shows the command `python3 main.py` being executed, resulting in the output:

```
mod2
mod1
rishabh@TNPL-rishabsagar:~/Downloads/pythona$
```

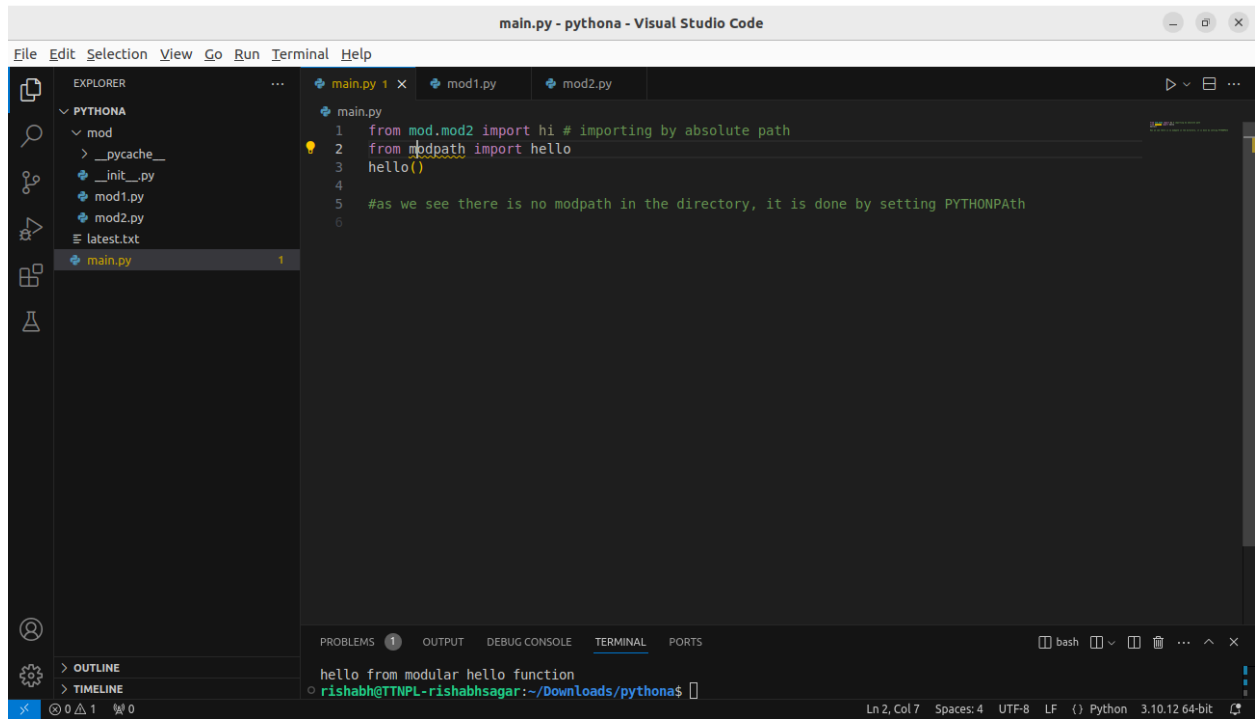


Proof that relative imports should be explicit-

Relative imports should be explicit because—

1. **Clarity**-In larger project clarity will be faded as the size of the project grows
2. **Ambiguity**- It leads to ambiguity as .. vs ... can lead to problems when the module structure changes or when scripts are executed from different directories.

3.Package Execution: Relative imports only work within a package context. If you attempt to run a script that contains relative imports directly Python will throw an error



The screenshot shows the Visual Studio Code interface with a Python project. The Explorer panel on the left shows a file structure with a `mod` directory containing `mod1.py` and `mod2.py`, and a `main.py` file in the root. The main editor displays `main.py` with the following code:

```
1 from mod.mod2 import hi # importing by absolute path
2 from modpath import hello
3 hello()
4
5 #as we see there is no modpath in the directory, it is done by setting PYTHONPATH
6
```

The terminal at the bottom shows the output of running the script:

```
hello from modular hello function
rishabh@TNPL-rishabhsagar:~/Downloads/pythona$
```

The status bar at the bottom indicates the file is `main.py` in the `pythona` workspace, using Python 3.10.12 64-bit.

The screenshot shows a Visual Studio Code window titled "main.py - pythona - Visual Studio Code". The Explorer sidebar on the left shows a project structure with a "PYTHONA" folder containing "mod" and "main.py". The main editor area displays a terminal with the following commands and output:

```
rishabh@TTNPL-rishabhsagar:~/Downloads/pythona$ export PYTHONPATH=/home/rishabh/Downloads/modular
rishabh@TTNPL-rishabhsagar:~/Downloads/pythona$ echo $PYTHONPATH
/home/rishabh/Downloads/modular
rishabh@TTNPL-rishabhsagar:~/Downloads/pythona$ python3 main.py
hello from modular hello function
rishabh@TTNPL-rishabhsagar:~/Downloads/pythona$
```

The screenshot shows a Visual Studio Code window titled "modpath.py - modular - Visual Studio Code". The Explorer sidebar on the left shows a project structure with a "MODULAR" folder containing "modpath.py" and "__init__.py". The main editor area displays the code for "modpath.py":

```
1 def hello():
2     print("hello from modular hello function")
3
```

A notification dialog is visible in the bottom right corner, stating: "A git repository was found in the parent folders of the workspace or the open file(s). Would you like to open the repository?" with buttons for "Yes", "Always", and "Never".

Q2) Explain the use of `from importlib import reload`

Ans. The reload method in the importlib is used to reload the module in real time i.e whenever we run the program which depend on any module, then that module is stored in the memory and whenever any changes happen in the module during execution it does not reflect, to reflect we have to stop the execution and re run the program hence reload will help to reflect the

changes immediately during the execution without re running the program. It is use in testing, web development etc

main.py - pythonc - Visual Studio Code

File Edit Selection View Go Run Terminal Help

EXPLORER

- PYTHONC
 - pkg
 - __pycache__
 - __init__.py
 - mod.py
 - main.py

main.py

```
1 from importlib import reload
2 import pkg.mod
3 import pkg
4
5 pkg.hi()# using method hi directly from the mod without doing pkg.mod.hi() beacuse we import
6 #it in the __init__
7 while True:
8     reload(pkg.mod)# remember to put module name in the reload and if in the package
9     #import the module and use generic way dont use from way
10    pkg.mod.hello()
11    input("Enter")
12
13
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

python3

```
○ rishabh@TNPL-rishabsagar:~/Downloads/pythonc$ python3 main.py
hi
Hello world
Hello india
Enter
Hello world
Enter
```

Ln 8, Col 53 Spaces: 4 UTF-8 LF () Python 3.10.12 64-bit

mod.py - pythonc - Visual Studio Code

File Edit Selection View Go Run Terminal Help

EXPLORER

- PYTHONC
 - pkg
 - __pycache__
 - __init__.py
 - mod.py
 - main.py

mod.py

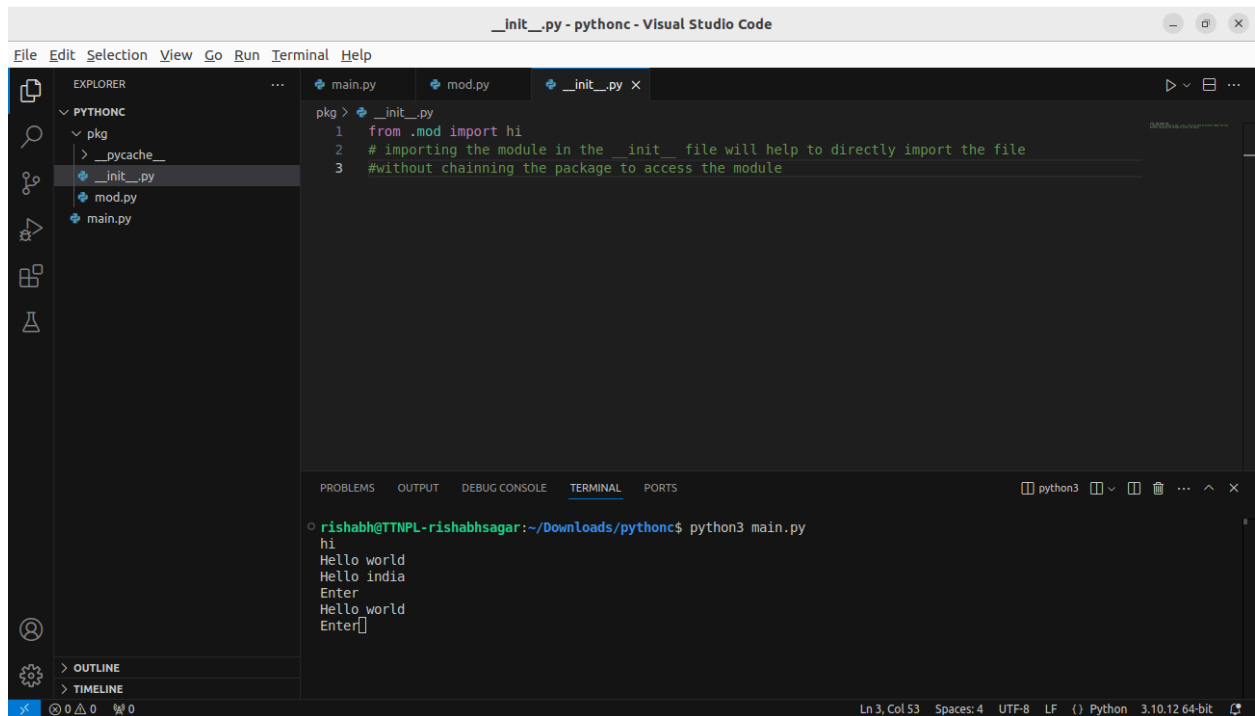
```
1 def hello():
2     print("Hello world")
3     # print("Hello india") commnet after running the program to see the changes
4
5 def hi():
6     print("hi")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

python3

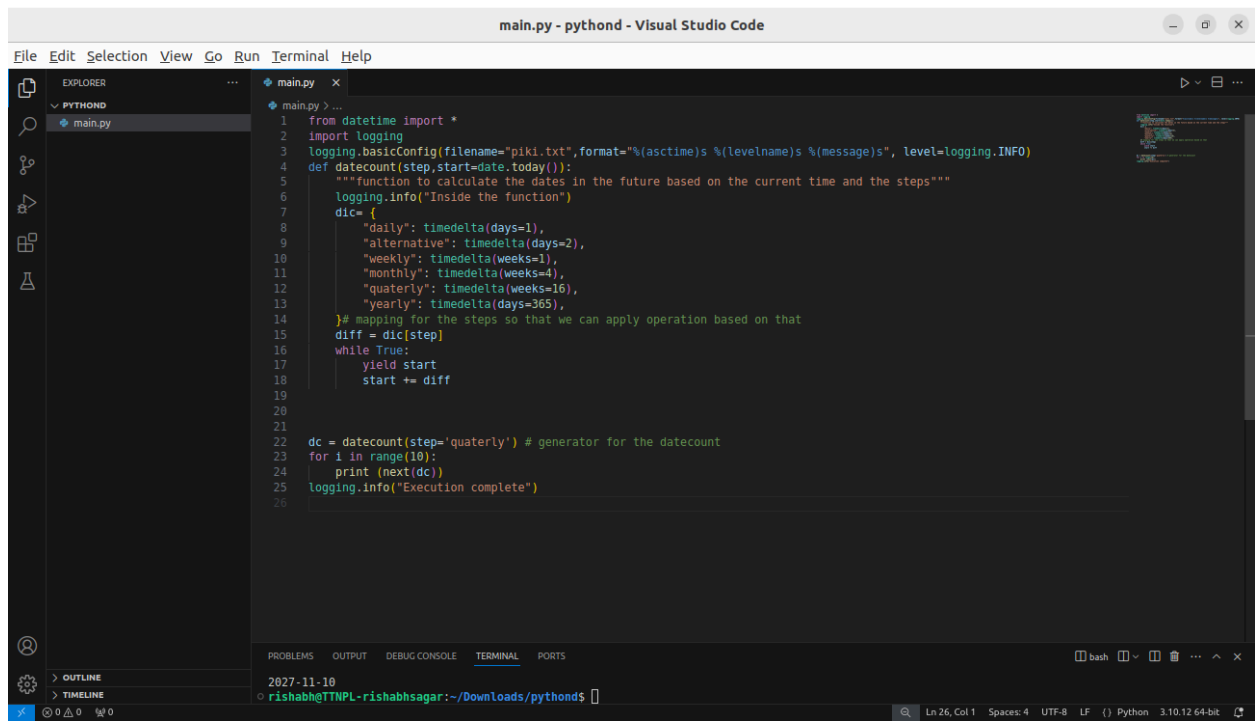
```
○ rishabh@TNPL-rishabsagar:~/Downloads/pythonc$ python3 main.py
hi
Hello world
Hello india
Enter
Hello world
Enter
```

Ln 6, Col 16 Spaces: 4 UTF-8 LF () Python 3.10.12 64-bit



Q3)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) example execution: `>> dc = datecount(step='weekly')` `>> for i in range(10): print (next(dc))` Output: 2025-01-17 2025-01-24 2025-01-31 2025-02-07 2025-02-14 2025-02-21 2025-02-28 2025-03-07 2025-03-14 2025-03-21

Ans.



```
main.py - pythond - Visual Studio Code
File Edit Selection View Go Run Terminal Help

EXPLORER
PYTHON
main.py

main.py
1 from datetime import *
2 import logging
3 logging.basicConfig(filename='piki.txt',format='%(asctime)s %(levelname)s %(message)s', level=logging.INFO)
4 def datecount(step,start=date.today()):
5     """function to calculate the dates in the future based on the current time and the steps"""
6     logging.info("Inside the function")
7     dic= {
8         "daily": timedelta(days=1),
9         "alternative": timedelta(days=2),
10        "weekly": timedelta(weeks=1),
11        "monthly": timedelta(weeks=4),
12        "quarterly": timedelta(weeks=16),
13        "yearly": timedelta(days=365),
14    }
15    )# mapping for the steps so that we can apply operation based on that
16    diff = dic[step]
17    while True:
18        yield start
19        start += diff
20
21
22 dc = datecount(step='quarterly') # generator for the datecount
23 for i in range(10):
24     print (next(dc))
25 logging.info("Execution complete")
26

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
2027-11-10
rishabh@TTNPL-riskabhsagar:~/Downloads/pythond$
```


main.py - pythond - Visual Studio Code

File Edit Selection View Go Run Terminal Help

EXPLORER

PYTHON

main.py

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

bash

```
• rishabh@TTNPL-rishabhsagar:~/Downloads/pythond$ python3 main.py
2025-02-05
2025-05-28
2025-09-17
2026-01-07
2026-04-29
2026-08-19
2026-12-09
2027-03-31
2027-07-21
2027-11-10
○ rishabh@TTNPL-rishabhsagar:~/Downloads/pythond$
```

OUTLINE

TIMELINE

Ln 26, Col 1 Spaces: 4 UTF-8 LF Python 3.10.12 64-bit

piki.txt - pythond - Visual Studio Code

File Edit Selection View Go Run Terminal Help

EXPLORER

main.py piki.txt

main.py

piki.txt

```
1 2025-02-05 22:27:02,509 INFO Inside the function
2 2025-02-05 22:27:02,509 INFO Execution complete
3
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

bash

```
2027-11-10
○ rishabh@TTNPL-rishabhsagar:~/Downloads/pythond$
```

OUTLINE

TIMELINE

Ln 3, Col 1 Spaces: 4 UTF-8 LF Plain Text