Modules and Packages

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

Absolute Imports : An absolute import state that the module is to be imported using its full path from the project's root folder.

Directory Structure :

```
├── Package1
│   ├── module1.py
│   ├── module3.py
│   └── sub_package
│       └── file1.py
```

 # module1.py

```
module1.py ×
Package1 > module1.py > package1_function
1   def package1_function():
2       print("This is the package 1 function")
```

 # sub_pacakge/file1.py

```
file1.py ×
Package1 > sub_package > file1.py > sub_module_function
1   def sub_module_function():
2       print("This is the sub module function")
```

 # module3.py

```
module3.py ×
Package1 > module3.py > ...
1   import module1
2   from sub_package import file1
3
4
5   def run():
6       module1.package1_function()
7       file1.sub_module_function()
8   run()
```

Output :

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

rohit@TTNPL-rohitvarshney:~/Modules and Packages$ /bin/python3 "/home/rohit/Modules and Packages/Package1/module3.py"
This is the package 1 function
This is the sub module function
rohit@TTNPL-rohitvarshney:~/Modules and Packages$
```

Relative Imports : Relative import specifies an object or module imported from its current location, that is the location where import statement resides.

Directory Structure :

```
├── sub_package
│   ├── file1.py
│   └── file2.py
└── sub_package2
    └── module2.py
```

 # sub_package/file2.py

```
file2.py   module2.py   module1.py
Package1 > sub_package > file2.py > ...
from ..sub_package2.module2 import sub_package2_function
def display():
    sub_package2_function()
display()
```

Output :

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

bash: /home/rohit/.bashrc: line 123: syntax error near unexpected token `unset'
bash: /home/rohit/.bashrc: line 123: `fi unset color_prompt force_color_prompt'
rohit@TTNPL-rohitvarshney:~/Modules and Packages$ python3 -m Package1.sub_package.file2
This is the sub_package 2 function
rohit@TTNPL-rohitvarshney:~/Modules and Packages$
```

PYTHONPATH : Storing the directory in the PYTHONPATH to make it read for pacakages.
It is used to set the path for the user-defined modules so that it can be directly imported into a Python program.

```
rohit@TTNPL-rohitvarshney:~$ export PYTHONPATH="/home/rohit/Modules and Packages/Package1"
rohit@TTNPL-rohitvarshney:~$ python3 -m module3
This is the package 1 function
This is the sub module function
rohit@TTNPL-rohitvarshney:~$
```

2) Explain the use of `from importlib import reload`.

- The `reload()` function from `importlib` is used to reload a module that we've already imported. This is useful when we make changes to a module's code and want to see those changes without restarting our Python session. Instead of re-importing the module, we just use `reload(module)` to refresh it and apply the new changes right away. It's especially helpful when we're working on code interactively.

```
file1.py   file2.py   iterat
file2.py
import file1
file1.show()

from importlib import reload
while True:
    file1.show()
    reload(file1)
```

Before Updating the module “file1.py”

```
file1.py   file2.py   iterat
file1.py > show
def show():
    print("Welcome TO THE NEW")
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

Welcome TO THE NEW
Welcome TO THE NEW
Welcome TO THE NEW
Welcome TO THE NEW
Welcome TO THE NEW
Welcome TO THE NEW
Welcome TO THE NEW
```

After Updating the module “file1.py”

```
file1.py   file2.py   iterat
file1.py > show
def show():
    print("Welcome TO THE NEW")
    print("Rohit Varshney")
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

Welcome TO THE NEW
Rohit Varshney
Welcome TO THE NEW
Rohit Varshney
Welcome TO THE NEW
Rohit Varshney
Welcome TO THE NEW
Rohit Varshney
Welcome TO THE NEW
Rohit Varshney
```

3) 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

```python
import datetime
def datecount(start, step):
    step = step.lower()
    while True:
        yield start

        if step == "daily":
            start += datetime.timedelta(days=1)
        elif step == "weekly":
            start += datetime.timedelta(weeks=1)
        elif step == "alternative":
            start += datetime.timedelta(days=2)
        elif step == "monthly":
            year = start.year
            month = start.month + 1
            if month > 12:
                month = 1
                year = year + 1
            start = start.replace(year=year, month=month)
        elif step == "quarterly":
            year = start.year
            month = start.month + 3
            if month > 12:
                month =month -12
                year =year + 1
            start = start.replace(year=year, month=month)
        elif step == "yearly":
            start = start.replace(year=start.year + 1)
        else:
            return "Invalid"

start = datetime.date.today()
step = input("Enter the step from the list ['alternative', 'daily', 'weekly', 'monthly', 'quarterly', 'yearly']:")
result = datecount(start, step)
for i in range(10):
    print(next(result))
```

```
/bin/python3 "/home/rohit/Modules and Packages/iterat"
bash: /home/rohit/.bashrc: line 123: syntax error near unexpected token `unset'
bash: /home/rohit/.bashrc: line 123: `fi unset color_prompt force_color_prompt'
rohit@TTNPL-rohitvarshney:~/Modules and Packages$ /bin/python3 "/home/rohit/Modules and Packages/iterat"
Enter the step from the list ['alternative', 'daily', 'weekly', 'monthly', 'quarterly', 'yearly']:weekly
2025-02-05
2025-02-12
2025-02-19
2025-02-26
2025-03-05
2025-03-12
2025-03-19
2025-03-26
2025-04-02
2025-04-09
rohit@TTNPL-rohitvarshney:~/Modules and Packages$
```

4) Write a `find.py` script which impleemnts Linux `find` command Implemnt below options: `-name` `-atime` `-type` `-maxdepth` Example use: To find all ".py" files (not folders) in home directory and 2 level sub-directories which where created recently in last 7 days write find.py ~/ -name "*.py" -type f -atime -7