Python Modules

🔹 What is a Module in Python?

A **module** in Python is a **file containing Python definitions and statements**. It may include **functions, classes, and variables**, and can also include runnable code. Modules help you **organize and reuse code** efficiently.

➤ Technical Definition:

A module is a Python object with arbitrarily named attributes that you can bind and reference. The module's name is the file name without the `.py` extension.

🔹 Why Use Modules?

Modules in Python help achieve:

- **Code Reusability** – Write once, use many times
- **Code Organization** – Keep codebase clean and understandable
- **Encapsulation** – Hide unnecessary implementation details
- **Maintainability** – Easier to update or debug a module rather than a monolithic script
- **Modularity** – Enables separation of concerns

🔹 Types of Modules

Type	Description	Example
Built-in Module	Predefined in Python standard library	`math`, `os`
User-defined	Created by the programmer	`my_module.py`
Third-party	External packages installed via `pip`	`numpy`, `flask`

🔹 How to Create and Use a Module

Step 1: Create a module (`my_module.py`)

python Copy code

```python
# my_module.py
def add(a, b):
    return a + b

def greet(name):
    return f"Hello, {name}!"
```

Step 2: Use it in another file

python Copy code

```python
# main.py
import my_module

print(my_module.add(3, 4))
print(my_module.greet("Alice"))
```

◆ Importing Modules – Techniques

Syntax	Example	Use Case
`import module`	`import math`	General usage
`import module as alias`	`import math as m`	Shorter names
`from module import func`	`from math import sqrt`	Use specific function
`from module import *`	`from math import *`	Imports all, but not recommended

◆ Module Search Path

When a module is imported, Python looks for it in the following order:

1. Current directory
2. Environment variable `PYTHONPATH`
3. Standard library directories

Check it using:

```python
import sys
print(sys.path)
```

🔹 `__name__ == "__main__"` Concept

Every Python module has a special built-in attribute `__name__`.

- If the module is **run directly**, `__name__` is `"__main__"`
- If it's **imported**, `__name__` is the module's name

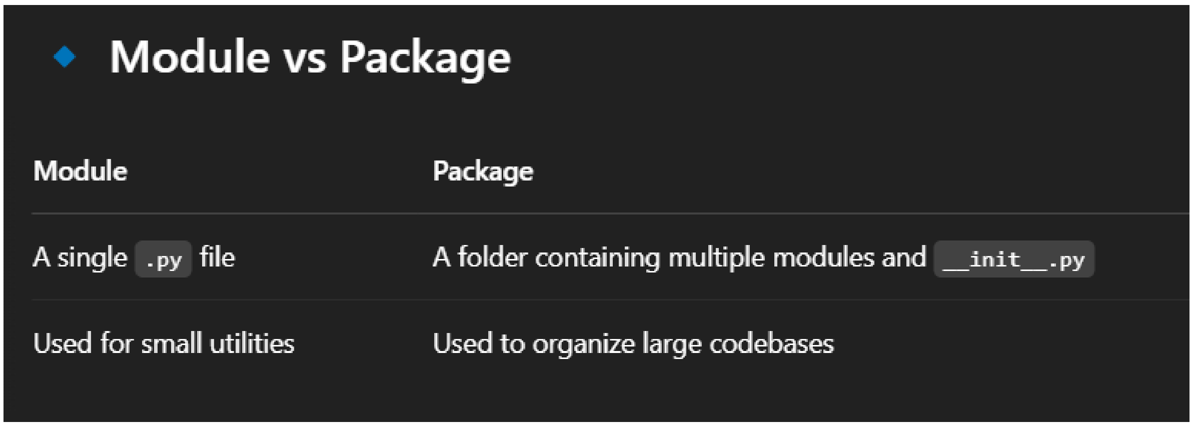
```python
# example_module.py
def run():
    print("Running module")

if __name__ == "__main__":
    run()
```

Useful for:

- Writing test cases
- Reusing code in other modules

🔹 Module vs Package


🔹 Module vs Package

Module	Package
A single `.py` file	A folder containing multiple modules and `__init__.py`
Used for small utilities	Used to organize large codebases

🔹 Popular Built-in Modules

Module	Use
`math`	Mathematical functions
`random`	Generate random numbers
`os`	Interact with operating system
`sys`	Access system-specific parameters
`datetime`	Work with date and time
`re`	Regular expressions
`json`	Parse and generate JSON

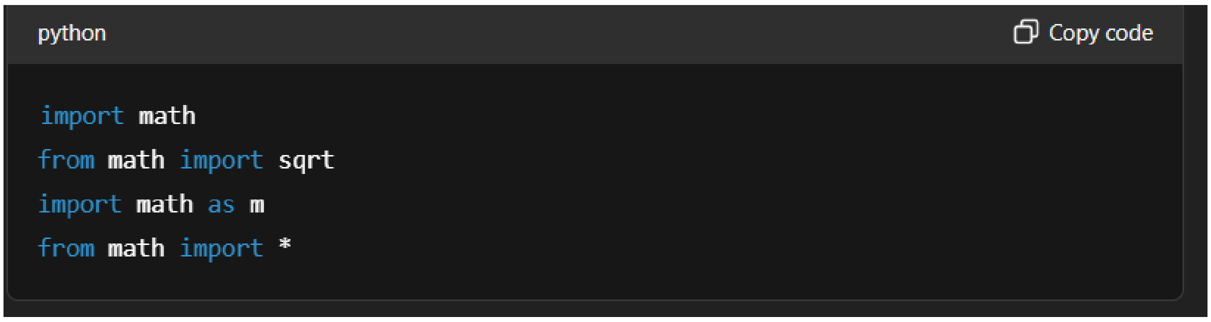
🎯 Python Module Interview Questions and Answers

Q1: What is a Python module?

Answer:
A module in Python is a file containing Python code, including functions, classes, and variables, that can be reused in other programs. It helps improve code reusability and organization.

Q2: How do you import a module in Python?

Answer: You can use different import styles:

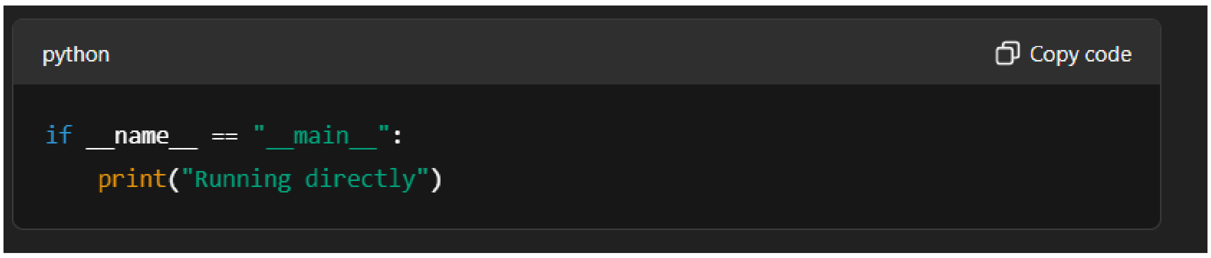


```python
import math
from math import sqrt
import math as m
from math import *
```

Q3: What is the purpose of `__name__ == "__main__"` in Python modules?

Answer:
It ensures that code inside this block runs **only when the module is executed directly**, not when imported.

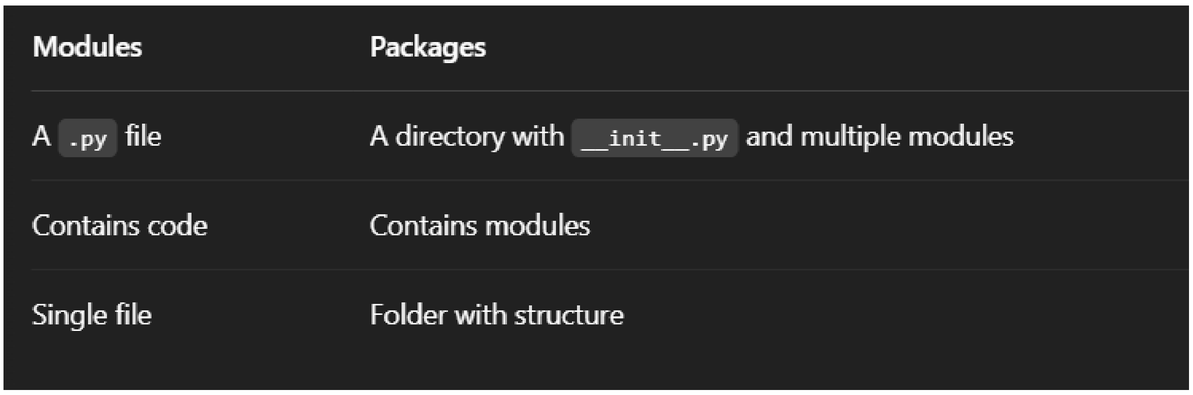
Example:



```python
if __name__ == "__main__":
    print("Running directly")
```

Q4: How are modules different from packages?

Answer:



Modules	Packages
A `.py` file	A directory with `__init__.py` and multiple modules
Contains code	Contains modules
Single file	Folder with structure

Q5: How does Python locate modules?

Answer: Python uses a list called `sys.path`, which contains:

- Current script directory

- PYTHONPATH
- Standard library paths

You can view it using:

```python
import sys
print(sys.path)
```

Q6: Can you name some important built-in modules?

Answer:

- `math` for math operations
- `os` for OS interaction
- `sys` for interpreter control
- `datetime` for date/time handling
- `random` for random numbers
- `re` for regex

Q7: What is a third-party module? How do you install one?

Answer: A third-party module is created by the community and not included in the standard library. You install it using:

```bash
pip install module_name
```

Example:

```bash
pip install requests
```

Then use it:

```python
import requests
```

Q8: Can a module be executed like a script?

Answer: Yes, if you include:

```python
if __name__ == "__main__"`:
    # Code here
```

It will run when you execute the file directly, not when imported.

Q9: How do you reload a module without restarting the program?

Answer: You can use the `importlib` module:

```python
import importlib
import my_module
importlib.reload(my_module)
```

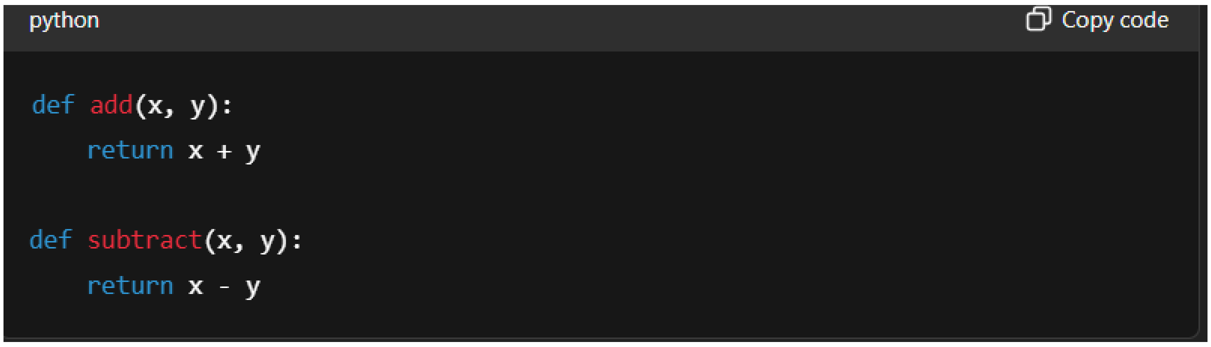
Q10: How do you create your own module?

Answer:

1. Create a Python file (e.g., `my_module.py`)
2. Define functions or variables
3. Import it using `import my_module` in another script

🧪 Sample Practice Module & Usage

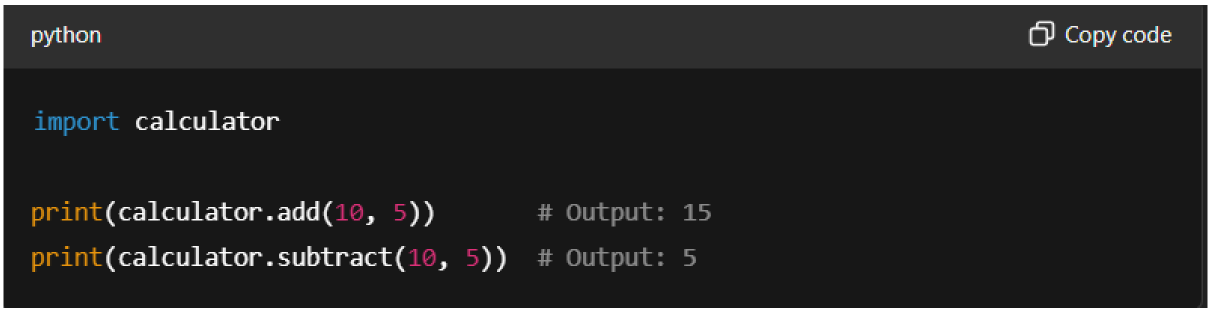
File: `calculator.py`



```python
def add(x, y):
    return x + y

def subtract(x, y):
    return x - y
```

File: `main.py`



```python
import calculator

print(calculator.add(10, 5))       # Output: 15
print(calculator.subtract(10, 5))  # Output: 5
```