1. What is a Module?

A **module** in Python is simply a file that contains Python definitions (functions, classes, variables) and executable statements.

- Technically, **any file with `.py` extension is a module**.
- Conceptually, it is a **logical unit of organization**: it allows us to group related functionalities together.

Why use modules?

- **Reusability**: Instead of writing the same code multiple times, you can import it wherever you need.
- **Maintainability**: Code is easier to read, maintain, and debug if separated into logical units.
- **Namespace management**: Modules help avoid name conflicts by keeping functions and variables encapsulated.

2. Types of Modules

Python modules can be classified into three main categories:

1. **Built-in Modules**
   - Already shipped with Python's standard library.
   - Examples: `math`, `os`, `sys`, `datetime`.
   - They cover common tasks such as mathematics, file handling, system operations, and more.
2. **Third-party Modules**
   - Developed by the Python community and installed via `pip`.
   - Examples: `numpy` (numerical computing), `pandas` (data analysis), `requests` (HTTP requests).
   - They drastically expand Python's capabilities.
3. **User-defined Modules**
   - Modules you create yourself by saving Python code in `.py` files.
   - Example: `my_utils.py` containing helper functions you often reuse.

3. Importing Modules

The `import` system is how Python allows you to access the functionality of modules.

- **Basic import**

python Kopyala Düzenle

```python
import math
print(math.sqrt(16))
```

- **Selective import**

python Kopyala Düzenle

```python
from math import sqrt
print(sqrt(16))
```

- **Alias**

python Kopyala Düzenle

```python
import numpy as np
```

- **Importing all (discouraged)**

python Kopyala Düzenle

```python
from math import *
```

4. What is a Package?

A **package** is a directory that contains multiple modules, often organized hierarchically.

- A folder containing an `__init__.py` file (in Python < 3.3) is recognized as a package.
- Modern Python allows **namespace packages** without `__init__.py`.

Example structure:

```markdown
mypackage/
    __init__.py
    module1.py
    module2.py
    subpackage/
        __init__.py
        module3.py
```

Usage:

```python
import mypackage.module1
from mypackage.subpackage import module3
```

5. Execution Context – `__name__ == "__main__"`

Every Python file has a special variable `__name__`.

- If the file is executed directly → `__name__ == "__main__"`.
- If the file is imported as a module → `__name__` is set to the module's filename.

This allows conditional execution:

```python
if __name__ == "__main__":
    main()
```

This ensures that the script's main functionality runs **only when executed directly**, not when imported.

6. How Python Finds Modules

When you import a module, Python searches in this order:

1. The current working directory.
2. Built-in library paths.
3. Paths listed in `sys.path`.
4. Installed site-packages (third-party libraries).

If the module is not found:
→ `ModuleNotFoundError`.

Tip: To inspect search paths:

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