Modules and pip

A module is a file of ready-made code you can import — learn the standard library, your own module files, and what pip install really does.

Introduction

You do not need to write everything yourself. Python comes with a large **standard library** — ready-made modules for maths, random numbers, dates, files, and much more. A **module** is simply a file of Python code that you can reuse with the word `import`. Instead of writing your own square-root code, you write `import math` and use `math.sqrt(16)`.

You can also write your own modules. Any Python file you save can be imported by another file in the same folder. That is how real projects stay organised: greeting code in one file, maths helpers in another, and a main file that imports both.

For code that is not built into Python, there is **pip**, the tool that installs extra packages from the public Python package collection. Pip does not come with magic — it downloads code written by other people, so you should install only packages you trust and actually need.

(Roman Urdu: Module aik tayyar code ki file hoti hai — import likh kar aap us ka code apnay program mein istemal kar saktay hain, baghair dobara likhay.)

Modules change how you think about code: your job is no longer to write everything, but to know what already exists and how to find it. Before writing a date or maths helper from scratch, check the standard library — `datetime`, `math`, `statistics` and `random` already solve most everyday needs, and they are tested by millions of users. Writing your own version is usually slower and buggier.

Key points

- **Import a module with `import`.** `import math` loads the maths module; then you use its tools with a dot: `math.sqrt(16)`.
- **Import one tool only with `from`.** `from math import sqrt` lets you write `sqrt(16)` without the `math.` prefix.
- **The standard library needs no install.** `math`, `random`, `datetime` and many more are already part of Python.
- **Your own file is a module too.** A file called `helpers.py` in the same folder can be imported with `import helpers`.
- **pip installs extra packages.** The command runs in your terminal, not inside your Python file: `pip install requests`.
- **Installing is not importing.** After installing a package, you still need `import requests` in your code before you can use it.

- **Only install what you trust and need.** Every extra package is someone else's code running in your project, and it can also break or become outdated.
- **Check what is installed with `pip list`.** It shows the packages in your current Python setup.
- **Module names must be valid Python names.** A file called `my helpers.py` (with a space) cannot be imported cleanly — use `my_helpers.py`.
- **`datetime` handles dates and times.** `datetime.date(2026, 10, 27)` builds a date, and `.strftime("%A")` prints its weekday name.
- **`statistics` summarises numbers.** `statistics.mean()` and `statistics.median()` give the average and the middle value of a list.

Examples

Example 1 — Using the standard library: math and random

```
import math
import random

print(math.sqrt(16))
print(math.floor(3.9))
print(math.ceil(3.1))
print(random.randint(1, 10))
print(random.choice(["apple", "banana", "mango"]))
```

What you will get (the random lines will vary each run):

```
4.0
3
4
7
banana
```

`math.sqrt(16)` is 4.0, `floor` rounds down and `ceil` rounds up. The random lines change on every run — that is the point of random numbers, and it also means you should not expect the exact same output twice.

Example 2 — Your own module file

Save this as a file called `helpers.py` in the same folder as your main program:

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

def double(number):
    return number * 2
```

Then, in a second file in the same folder, write:

```
import helpers

print(helpers.greet("Ali"))
print(helpers.double(21))
```

What you will get:

```
Hello Ali!
42
```

The second file does not contain the greeting or doubling code — it borrows both from `helpers.py` through the import. Change `helpers.py` once, and every file that imports it gets the new version.

Example 3 — pip as a concept (terminal command, not Python code)

This example is not Python code — it is a terminal command. In your terminal, you would type:

```
pip install requests
```

What it does: it downloads the `requests` package (a popular tool for fetching web pages from Python code) and installs it into your Python setup. It does not mean the package is already on your computer, and it does not run anything by itself.

After a successful install, your Python code could then start with:

```
import requests
```

Do not run the install unless you actually need the package for a project. For the lessons in this series, the standard library is enough — this example is here so you recognise the command when you meet it later.

Example 4 — Dates with datetime

Real programs constantly need dates: today's date, a deadline, or which weekday a date falls on. The `datetime` module handles all of it:

```
import datetime
today = datetime.date(2026, 10, 27)
print(today)
print(today.year)
print(today.strftime("%A"))
```

What you will get:

```
2026-10-27
2026
Tuesday
```

Dates print in year-month-day order. `strftime("%A")` formats the date as its full weekday name — 27 October 2026 is a Tuesday. Try changing the day to 28 and see the weekday change to Wednesday.

Example 5 — Averages with statistics

A teacher with a list of marks wants the average and the middle value. The `statistics` module gives both in one line each — no hand-written loops needed:

```
import statistics
marks = [80, 90, 70, 85]
print(statistics.mean(marks))
print(statistics.median(marks))
```

What you will get:

```
81.25
82.5
```

The mean is (80 + 90 + 70 + 85) / 4 = 81.25. The median sorts the marks (70, 80, 85, 90) and takes the middle — with an even count, it averages the two middle values, (80 + 85) / 2 = 82.5.

Example 6 — Real-world case: rounding a price and stamping today's date

A small shop rounds prices up to the nearest hundred for a printed price list, and stamps each list with today's date. Both jobs are one module call each:

```
import datetime
import math
price = 999
print("Rounded up:", math.ceil(price / 100) * 100)
today = datetime.date.today()
print("Today:", today)
```

What you will get (the date line shows the day you run it):

```
Rounded up: 1000
Today: <today's date>
```

`math.ceil(999 / 100)` rounds 9.99 up to 10, and 10 × 100 = 1000. `date.today()` always returns the real current date, so its output changes every day — that is correct behaviour, not an error.

Quick recap

- A module is reusable Python code, imported with `import`.
- The standard library (`math`, `random`) is built in — no install needed.
- Your own `.py` file in the same folder is importable as a module.
- pip installs extra packages from the terminal: `pip install <name>`.
- Installing and importing are two separate steps.
- `datetime.date` builds a date; `strftime("%A")` gives its weekday.
- `statistics.mean()` and `statistics.median()` summarise a list of numbers.

Common mistakes

1. **Writing `pip install` inside a Python file** — pip is a terminal command, not Python code, so it errors inside a `.py` file. Fix: type it in the terminal, then write `import <name>` in your code.
2. **Forgetting to import before using** — `math.sqrt(16)` without `import math` gives a name error. Fix: put all imports at the top of your file.
3. **Naming your file the same as a standard module** — a file called `math.py` will shadow the real `math` module and confuse every import. Fix: never name your files `math.py`, `random.py`, `test.py` or similar standard names.
4. **Importing a module from a different folder** — Python looks in the same folder and its standard paths, not everywhere on your computer. Fix: keep the module file in the same folder as the file that imports it.
5. **Installing packages you do not need** — a long install list makes a project fragile and hard to share. Fix: install only what the project actually uses, and note each one down.
1. **Importing everything with `from math import *`** — it dumps dozens of names into your file, where they can silently hide your own variables. Fix: import the module, or name exactly what you need: `from math import sqrt`.
2. **Two files importing each other** — if `a.py` imports `b.py` and `b.py` imports `a.py`, Python goes in a circle and fails. Fix: put the shared code in a third file both can import.

Practice

1. Write a program that uses `math` to print the square root of 25, and `random` to pick a random name from a list of three friends. Run it twice and note what changes.
2. Create a file `helpers.py` with a function that returns the square of a number. In a second file, import it and print the square of 9. Check that the answer is 81.
3. Open your terminal and run `pip list`. Write down three package names you see — do not install anything new for this task.
4. Write, in your own words in two lines, the difference between installing a package and importing it.
5. Use `datetime` to print the current year and month, and to build the date of your birthday this year and print its weekday with `strftime("%A")`.

6. Use `statistics` to find the mean and median of `[60, 75, 80, 90, 95]`. Check the mean by hand (answer: 80).
7. Create a module file `shapes.py` with two functions, `area_square(side)` and `area_circle(radius)` (use `math.pi` for the circle). In a second file, import it and print both areas.

Practice quiz

1. What does import math do?

- Installs Python
- Brings the math module's ready-made code into your program
- Creates a new file
- Deletes maths code

Answer: Brings the math module's ready-made code into your program

2. What is pip used for?

- Running Python programs
- Installing extra Python packages from the internet
- Printing files
- Fixing errors automatically

Answer: Installing extra Python packages from the internet

3. How can your own file become a module?

- Rename it to .txt
- Any .py file can be imported by another program
- Only files in the Python folder count
- Modules must be bought

Answer: Any .py file can be imported by another program

Watch the video lesson: https://youtu.be/8XK5_VkqLBI

Read online: https://scopetreasure.com/notes/py-modules-pip · More free tools and notes at https://scopetreasure.com