9. Modules

❖Introduction to Python modules and importing modules.

What Is a Python Module?

A module is simply a .py file that contains Python definitions—functions, classes, variables, and executable statements. You can think of it as a reusable code library. Grouping related functionality into modules makes your code cleaner, more maintainable, and easier to share.

Types of Modules

Python supports various module types:

- Built-in (compiled in C, e.g. sys, math)
- C extensions (shared libraries)
- Python source (.py files) and compiled bytecode (.pyc)
- Packages (directories containing __init__.py)
- Namespace packages (PEP 420, no __init__.py)

Syntax :

```
import b
```

b.fun()

❖Standard library modules: math, random

- **The math Module**

  The math module offers a host of functions and constants highly useful for numerical and scientific tasks:

  - Common constants: math.pi, math.e, etc.
  - Number-theoretic functions: factorial(), gcd(), isqrt(), comb(), perm()
  - Floating-point operations: ceil(), floor(), fabs(), trunc(), fmod(), frexp(), copysign()
  - Power and logs: pow(), exp(), log(), log2(), log10()
  - Trig/hyperbolic functions: sin(), cos(), tan(), sinh(), cosh(), etc.
  - Special functions: erf(), gamma(), lgamma()
    These functions work on real numbers; for complex inputs, use cmath instead

- **The random Module**

This module implements **pseudo-random** number generation through the widely-used **Mersenne Twister algorithm** and offers a flexible API for randomness

Core functions:

- random.random(): returns a float in [0.0, 1.0)
- randint(a, b): random integer N such that a ≤ N ≤ b
- randrange(): pick a value from a range (like range(start, stop, step))
- choice(seq): randomly pick from a sequence
- shuffle(list): randomly permute a list in place
- sample(seq, k): choose k unique items
- Distribution functions: uniform(), normalvariate(), gauss(), lognormvariate(), expovariate(), betavariate()

❖Creating custom modules.

Import Mechanism: Find → Load

1. **Check cache**: Looks in sys.modules. If found, returns existing module object.

2. **Finder stage**: Traverses sys.meta_path and then sys.path with registered finders to select a loader
3. **Loader stage**: The chosen loader reads the code (e.g., .py file), compiles it (if needed), and executes it in a newly created module object.
4. **Cache**: Stores the module object in sys.modules to avoid reloading next time .

Optional: You can bypass cache and force re-execution using importlib.reload() .

Step : 1

```
# circle_area.py
def area_of_circle(r):
    return 3.14159 * r * r
coolpy = "LearnPython.com is cool!"
```

Step : 2

```
import circle_area


print(circle_area.area_of_circle(7))
print(circle_area.coolpy)
```