Python Overview

Presenter : Pandya Shaunak Rajnikant

Date : 15th July 2025

What Is Python?

- Python is a high-level, interpreted programming language designed for readability and rapid development.
- Clean, expressive syntax minimizes boilerplate
- Supports procedural, object-oriented, and functional paradigms
- Extensive “batteries-included” standard library
- Open-source with a vibrant global community

Origins & Evolution

- Python’s journey began in the late 1980s under Guido van Rossum at CWI in the Netherlands.
- 1989: Guido starts the Python project to improve on ABC language
- 1991: First public release (Python 0.9.0) with functions, modules, exceptions
- 2000: Python 2.0 adds list comprehensions and garbage collection
- 2008: Python 3.0 introduces breaking changes for cleaner syntax
- 2023: Python 3.11/3.12 focus on performance enhancements and typing

Why Python Gained Popularity

- Several factors fueled Python’s rapid adoption across domains:
- Readable code accelerates learning and maintenance
- Rich standard library spans web, data, networking, automation
- Strong foothold in data science, machine learning, web frameworks
- Beginner-friendly while scalable for large, complex systems

Python Functions

- Functions are reusable blocks of code that encapsulate logic and improve organization.
- Defined with the def keyword
- Can accept positional, keyword, default, *args, and **kwargs parameters
- Return any Python object (or None by default)
- Docstrings document purpose and usage

Advanced Function Concepts

- Unlock more flexibility and power with advanced patterns:
- Default arguments simplify common use cases
- *args and **kwargs collect variable argument lists
- Lambda functions for concise, anonymous expressions
- Decorators for modifying or extending behavior without altering the original function

Python Modules

Modules are single .py files that group related code into namespaces.
•Import with import module_name or from module_name import name
•Types of modules:
•Built-in (e.g., os, sys)
•Standard Library (e.g., json, dataclasses)
•Third-Party (e.g., requests, numpy)
•Custom (project-specific utilities)

Module Type	Example	Purpose
Built-in	os	System interactions
Standard Lib	json	JSON parsing & serialization
Third-Party	requests	HTTP client library
Custom	mymodule	Your application’s helpers

Creating & Using Custom Modules

- Steps to build and utilize your own Python modules:
- Create a file named mymodule.py alongside your scripts
- Define functions, classes, or constants inside it
- In another script, import:

Packages & Best Practices

- Organize code at scale and follow conventions for maintainability:
- Packages are directories with an __init__.py file, enabling hierarchical namespaces
- Use virtual environments (venv, conda) to isolate dependencies
- Manage libraries via pip and requirements.txt
- Adhere to PEP 8 for naming, formatting, and code style
- Keep functions small and focused; name modules clearly

Thank You