



TECH NEXTGEN ACADEMY

CORE PYTHON PROGRAMMING

Think. Code. Automate. Solve. Build.

Foundation to Professional Level | Practical Classroom Training | Project-Based Learning

Duration: 5–6 Months | **Prerequisites:** Basic computer knowledge. No prior programming or advanced math required. | **Goal:** Programming Beginner → Confident Python Programmer → Strong Foundation for Professional Development

BUILD A STRONG PROGRAMMING FOUNDATION WITH PYTHON

Python is one of the world's most popular and versatile programming languages, widely used in software development, automation, scripting, backend development, data analysis, testing, AI, and application development. Its simple syntax and powerful built-in features make Python an excellent language for beginners, while its flexibility allows programmers to build sophisticated real-world applications.

At Tech NextGen Academy, Core Python Programming is designed to take you from complete beginner to confident Python programmer through structured concepts, hands-on coding, logical problem-solving, debugging challenges, and real-world projects. The course focuses not only on learning Python syntax, but on developing the ability to think logically, write clean code, work with data, automate tasks, and build practical applications.

WHO IS THIS COURSE FOR?

-  School & College Students
-  BCA / B.Sc. / B.Tech Students
-  Computer Science & Engineering Students
-  Commerce & Non-IT Students
-  Job Seekers & Freshers
-  Aspiring Software Developers
-  Beginners in Programming
-  Students Switching to IT
-  Students Preparing for Coding Interviews
-  Students Interested in Automation & AI
-  Students Planning to Learn Data Analytics
-  Students Planning to Learn Backend Development
-  Anyone who wants to learn Python from the ground up

WHAT YOU WILL MASTER

MODULE 01

Programming Fundamentals & Logical Thinking

Learn to Think Before You Code

Topics: What is programming? | What is a programming language? | Why Python? | Features of Python | Advantages of Python | Applications of Python | Python's role in modern software development | Program development process | Source code & execution | Interpreter concepts | Algorithms | Flowcharts | Pseudocode | Problem decomposition | Input → Process → Output | Logical thinking | Problem-solving strategies

PRACTICAL: Convert real-world problems into: Problem → Algorithm → Flowchart → Python Program → Output.

MODULE 02

Python Installation & Development Environment

Set Up Your Python Development Workspace

Topics: Python installation | Python versions | Python interpreter | Python Interactive Shell | .py files | Running Python programs | Command-line execution | IDE introduction | Code editors | Integrated terminal | Python execution workflow | Comments | Keywords | Identifiers | Naming conventions | Basic debugging

Development Workflow: Write → Run → Test → Debug → Improve

PRACTICAL: Create and execute your first Python programs using different development environments.

MODULE 03

Variables, Data Types & Operators

Build the Core of Your Python Programs

Variables: What is a variable? | Variable creation | Assignment | Multiple assignment | Naming rules | Naming conventions | Dynamic typing | Variable reassignment | Scope basics

Built-in Data Types: int | float | complex | bool | str | list | tuple | set | dict | None

Operators: Arithmetic operators | Comparison operators | Logical operators | Assignment operators | Membership operators | Identity operators | Bitwise operators | Operator precedence | Associativity

Type Conversion: int(), float(), str(), bool() | Explicit type conversion | Type checking using type()

PRACTICAL: Calculator, Marks calculator, Salary calculator, Area calculator, Currency conversion program, Simple billing application.

MODULE 04

Input, Output & String Fundamentals

Make Python Programs Interactive

Input & Output: input() | print() | Multiple inputs | Formatted output | Escape sequences | Separator and end parameters | Basic input validation

String Fundamentals: What is a string? | String creation | Single and double quotes | Multi-line strings | String indexing | Negative indexing | String slicing | String concatenation | String repetition | String immutability

String Methods: lower(), upper(), title(), capitalize(), strip(), replace(), split(), join(), find(), count(), startswith(), endswith()

String Formatting: String concatenation | % formatting | str.format() | f-strings

PRACTICAL: Student information system, Employee information program, Username generator, Text-processing application, Simple billing program.

MODULE 05

Decision Making & Conditional Statements

Make Programs Think & Decide

Concepts: if | if-else | elif | Nested conditions | Multiple conditions | Logical expressions | Conditional expressions | Truthy and falsy values | Comparison operators | Boolean logic

Problem-Solving: Even/Odd checker | Positive/Negative checker | Largest of numbers | Smallest of numbers | Grade calculator | Eligibility checker | Electricity bill calculation | Salary calculation | Discount calculator | Tax calculation | Menu-driven programs

CHALLENGE: Solve real-world decision-making problems independently using Python logic.

MODULE 06

Loops & Iteration

Automate Repetitive Tasks

Concepts: Why loops? | for loop | while loop | range() | Nested loops | break | continue | pass | Infinite loops | Loop tracing | Counter-controlled loops | Condition-controlled loops

Programming Practice: Number patterns | Character patterns | Multiplication tables | Factorial | Fibonacci series | Prime numbers | Armstrong numbers | Palindrome numbers | Reverse numbers | Sum of digits | Number counting

PRACTICAL: Build pattern generators and mathematical programs using nested loops.

MODULE 07

Functions & Modular Programming

Write Reusable & Organized Code

Concepts: What is a function? | Advantages of functions | Function definition | Function calling | Parameters | Arguments | Return values | return | None | Positional arguments | Keyword arguments | Default arguments | Variable-length arguments (*args, **kwargs) | Function scope (Local variables, Global variables, global keyword) | Modular programming

FUNCTION DESIGN: Learn to break large problems into smaller, reusable functions.

MODULE 08

Lists & Sequence Data

Master One of Python's Most Important Data Structures

Fundamentals: What is a list? | Creating lists | List indexing | Negative indexing | List slicing | Adding elements | Updating elements | Removing elements | Iterating through lists | Nested lists | Lists of objects/data

List Methods: append(), extend(), insert(), remove(), pop(), clear(), index(), count(), sort(), reverse(), copy()

List Operations & Comprehensions: Searching | Sorting | Filtering | Counting | Finding maximum/minimum | Calculating totals | Removing duplicates | Basic list comprehensions | Conditional comprehensions | Nested comprehensions | Practical use cases

PRACTICAL PROJECT: Build a Student Marks & Performance Analysis System using lists.

MODULE 09

Tuples, Sets & Dictionaries

Master Python's Core Collections

Tuples: Tuple fundamentals | Creating tuples | Indexing | Slicing | Tuple unpacking | Immutable sequences | Tuple methods

Sets: Set fundamentals | Creating sets | Adding elements | Removing elements | Unique values | Set operations (Union, Intersection, Difference, Symmetric difference) | Membership testing

Dictionaries: Dictionary fundamentals | Key-value pairs | Creating dictionaries | Accessing values | Adding data | Updating data | Removing data | Nested dictionaries | Iterating through dictionaries | Methods: keys(), values(), items(), get(), update(), pop(), popitem(), setdefault()

PRACTICAL: Build Student → Employee → Product → Inventory data management applications using Python collections.

MODULE 10

Advanced Data Handling & Comprehensions

Write Cleaner & More Powerful Python Code

Concepts: Nested lists | Nested dictionaries | List of dictionaries | Dictionary of lists | Dictionary of dictionaries | Iterating nested data | Data transformation | Filtering collections | Sorting collections | Custom sorting using key | sorted() | reversed()

Comprehensions & Unpacking: List comprehension | Set comprehension | Dictionary comprehension | Conditional comprehensions | Nested comprehensions | Multiple assignment | Sequence unpacking | * unpacking | ** dictionary unpacking

PRACTICAL: Process real-world datasets such as Student records, Employee records, Product inventory, Sales information.

MODULE 11

Object-Oriented Programming Fundamentals

Enter the World of Professional Python Development

OOP Fundamentals: Procedural vs Object-Oriented Programming | What is an object? | What is a class? | Class attributes | Instance attributes | Methods | Object creation | self | Accessing attributes | Instance methods | Class methods | Static methods

Real-World Modeling: Real-World Entity → Class → Object → Properties → Behaviors

PRACTICAL: Create classes for Student, Employee, Bank Account, Product, Customer, Library Book.

MODULE 12

Constructors, Encapsulation & Abstraction

Design Better Python Classes

Constructors: __init__() | Constructor parameters | Object initialization | Default values | Multiple-object creation

Encapsulation: Public attributes | Protected convention | Private convention | Name mangling | Getter methods | Setter methods | Controlled data access

Abstraction: What is abstraction? | Why abstraction matters | Abstract classes | Abstract methods | abc module | Interfaces-like design in Python

PRACTICAL: Build a secure Bank Account → Customer → Account Operations application using classes, encapsulation and abstraction.

MODULE 13

Inheritance & Code Reusability

Build Relationships Between Classes

Concepts: What is inheritance? | Parent class | Child class | Base class | Derived class | Single inheritance | Multilevel inheritance | Multiple inheritance | Hierarchical inheritance | Method overriding | super() | Constructor inheritance | Method reuse

Real-World Examples: Employee → Manager | Vehicle → Car | Person → Student | Account → Savings Account

PRACTICAL: Build a class hierarchy for a real-world business application.

MODULE 14

Polymorphism & Advanced OOP

One Interface → Multiple Behaviors

Polymorphism: What is polymorphism? | Method overriding | Duck typing | Dynamic behavior | Polymorphic functions | Polymorphism through inheritance | Operator overloading

Special Methods: `__str__()` | `__repr__()` | `__len__()` | `__eq__()` | `__lt__()` | `__add__()`

Object Relationships: Association | Aggregation | Composition | Inheritance vs composition

PRACTICAL: Build a polymorphic application using different types of real-world objects.

MODULE 15

Modules, Packages & Code Organization

Organize Python Applications Professionally

Modules & Packages: What is a module? | Creating modules | Importing modules (`import`, `from...import`) | Import aliases | Built-in modules | User-defined modules | Package fundamentals | Package structure | Creating packages | Importing from packages | `__init__.py` | Organizing large applications

Standard Library Modules: `math` | `random` | `datetime` | `os` | `sys` | `statistics` | `json` | `re`

PRACTICAL: Create a multi-module Python application with properly organized files.

MODULE 16

Exception Handling & Error Management

Build Reliable Python Applications

Concepts: What is an exception? | Syntax errors vs runtime exceptions | Exception handling concepts (`try`, `except`, `else`, `finally`) | Multiple exceptions | Exception hierarchy | `raise` | Custom exceptions | Exception propagation | Error-handling strategies

Common Exceptions: `ValueError` | `TypeError` | `IndexError` | `KeyError` | `NameError` | `ZeroDivisionError` | `FileNotFoundError` | `AttributeError`

PRACTICAL: Build applications that safely handle invalid input, division by zero, missing records, invalid menu choices, file errors, incorrect data types.

MODULE 17

File Handling & Data Persistence

Store & Retrieve Data Permanently

File Handling: File paths | Opening files | Closing files | Reading files | Writing files | Appending data | File modes | Text files | Binary files | Context managers (with statement) | File Methods: `read()`, `readline()`, `readlines()`, `write()`, `writelines()`, `seek()`, `tell()`

Working with CSV & JSON: CSV fundamentals | Reading/Writing CSV files | `csv` module | Processing tabular data | JSON fundamentals | Reading/Writing JSON | Serialization | Deserialization | `json` module

PRACTICAL: Build a file-based Student / Employee / Inventory Management System with persistent data storage.

MODULE 18

Regular Expressions & Text Processing

Learn to Search, Validate & Extract Text

Regex Concepts: What are regular expressions? | Pattern matching | Character classes | Special characters | Quantifiers | Anchors | Groups | Capturing groups | Alternation | Raw strings

re Module: search(), match(), fullmatch(), findall(), finditer(), split(), sub()

Real-World Validation: Email validation, Phone number validation, Password validation, Postal code validation, Data extraction

PRACTICAL PROJECT: Build a Text & Data Validation Tool using regular expressions.

MODULE 19

Iterators, Generators & Functional Python

Write More Efficient Python Code

Iterators & Generators: Iterable vs iterator | iter() | next() | Custom iterators | Iteration protocol | Generator fundamentals | yield | Generator functions | Generator expressions | Lazy evaluation | Memory-efficient processing

Functional Programming: First-class functions | Functions as arguments | Functions as return values | Lambda expressions | map() | filter() | reduce() | zip() | enumerate()

PRACTICAL: Process large collections and repetitive data efficiently using iterators, generators and functional techniques.

MODULE 20

Decorators, Context Managers & Advanced Python Concepts

Move Toward Professional Python Programming

Decorators & Context Managers: What is a decorator? | Function decorators | Nested functions | Wrapper functions | @ syntax | Practical decorator use cases | Context manager fundamentals | with statement | Resource management | Custom context managers

Advanced Concepts: Scope & namespaces (LEGB rule) | Mutable vs immutable objects | Object identity (is vs ==) | Shallow vs deep copy (copy module) | *args and **kwargs | Unpacking techniques

PRACTICAL: Create reusable utilities using decorators and context-management concepts.

MODULE 21

Debugging, Testing & Code Quality

Don't Just Write Code. Learn to Improve It.

Error Types & Debugging: Syntax errors | Runtime exceptions | Logical errors | Input-related errors | File-related errors | Reading error messages | Tracebacks | Breakpoints | Step-by-step execution | Variable inspection | Debugging techniques | Finding logical mistakes | Testing individual functions

Code Quality & Testing: Meaningful naming | Proper indentation | Clean functions | Modular code | Reusability | Avoiding unnecessary duplication | Documentation | Comments | Basic PEP 8 awareness | Readable code | Maintainable code | Test cases | Positive testing | Negative testing | Boundary testing | Functional testing basics | assert | Unit testing concepts | unittest

PRACTICAL: Debug intentionally broken Python programs and prepare a basic test report.

MODULE 22

Problem-Solving & Python Coding Challenges

Turn Python Knowledge into Real Programming Skills

Students solve progressively difficult problems covering: Number problems | Pattern problems | Mathematical logic | Strings | Lists | Tuples | Sets | Dictionaries | Functions | Comprehensions | Searching | Sorting | File handling | JSON | Regular expressions | Object-Oriented Programming | Exception handling | Iterators | Generators

Coding Challenge Levels: Beginner → Intermediate → Advanced

Execution Flow: Understand → Plan → Code → Test → Debug → Improve



MODULE 23 — REAL-WORLD CORE PYTHON APPLICATION DEVELOPMENT

Students work on a complete application from requirement analysis to final presentation following 13 systematic steps:

- 1 **Requirement Analysis:** Understand the business problem and identify required features.
- 2 **Problem Design:** Identify data, operations and application requirements.
- 3 **Program Structure:** Divide the application into logical functions and modules.
- 4 **Data Management:** Apply Lists → Tuples → Sets → Dictionaries.
- 5 **OOP Implementation:** Apply Classes → Objects → Encapsulation → Inheritance → Polymorphism.
- 6 **Error Management:** Implement Validation → Exceptions → Custom Error Handling.
- 7 **Data Storage:** Implement Text Files → CSV → JSON.
- 8 **Text Processing:** Apply string operations and regular expressions where required.
- 9 **Code Organization:** Use Functions → Modules → Packages.
- 10 **Testing:** Test different inputs and application scenarios.
- 11 **Debugging:** Identify and correct programming errors.
- 12 **Optimization:** Improve code readability, organization and efficiency.
- 13 **Presentation:** Explain the architecture, logic, features and implementation.



CAPSTONE PROJECT — COMPLETE PYTHON MANAGEMENT SYSTEM

Students will develop a complete menu-driven Python application using the concepts learned throughout the course.

Project Options (Choose One):

Student Management System • Employee Management System • Library Management System • Banking Management System • Inventory Management System • Hospital Management System • Hotel Management System • Billing & Sales Management System • Expense Management System

Project Features:

User-friendly menu • Modular program structure • Functions • Lists, Dictionaries, Sets, Tuples • Classes & objects • Encapsulation, Inheritance, Polymorphism • Input validation & Exception handling • Search, Sorting, Filtering functionality • Reports & persistent File/CSV/JSON storage • Regular-expression validation • CRUD operations (Add, View, Search, Update, Delete records) • Summaries generation

Final Project Flow: Requirement → Design → Program Structure → Coding → Data Management → OOP → Validation → Exception Handling → File/JSON Storage → Testing → Debugging → Optimization → Presentation

Final Deliverables:

- 📁 Complete Python Source Code
- 📄 Requirement & Project Documentation
- 🛠️ Program / Module Structure
- 🧪 Test Cases
- 🐛 Debugging Record
- 💾 Persistent Data Storage

TECH NEXTGEN LEARNING METHOD & JOURNEY

LEARN → THINK → CODE → TEST → DEBUG → BUILD → PRESENT → GROW

Your Learning Journey:

FOUNDATION (Programming Concepts & Logical Thinking) → ENVIRONMENT (Python Installation & Tools) → CORE (Variables, Types, Operators) → INTERACTION (Input, Output, Strings) → LOGIC (Conditionals) → AUTOMATION (Loops) → MODULARITY (Functions) → DATA (Lists) → COLLECTIONS (Tuples, Sets, Dictionaries) → DATA PROCESSING (Comprehensions) → OOP FOUNDATION (Classes & Objects) → CLASS DESIGN (Constructors, Encapsulation, Abstraction) → REUSABILITY (Inheritance) → FLEXIBILITY (Polymorphism) → ORGANIZATION (Modules & Packages) → RELIABILITY (Exception Handling) → STORAGE (Files, CSV, JSON) → TEXT PROCESSING (Regex) → ADVANCED PYTHON (Iterators, Generators, Functional) → PROFESSIONAL PYTHON (Decorators, Context Managers) → QUALITY (Debugging & Testing) → PROBLEM-SOLVING (Challenges) → APPLICATION (Real-World Project) → ACHIEVEMENT (Capstone + Assessment + Certification)

WHAT YOU WILL BE ABLE TO DO

- ✓ Understand programming fundamentals & Python's role
- ✓ Install, configure, and execute Python programs
- ✓ Work with variables, built-in data types & operators
- ✓ Process user input, format output & manipulate strings
- ✓ Implement conditional logic, loops & pattern problems
- ✓ Design reusable functions with positional/keyword/*args/**kwargs
- ✓ Work with Lists, Tuples, Sets, Dictionaries & comprehensions
- ✓ Apply OOP (Classes, Encapsulation, Abstraction, Inheritance, Polymorphism)
- ✓ Organize code into Modules and Packages
- ✓ Handle errors with exception handling & custom exceptions
- ✓ Read/Write Text, CSV, and JSON files
- ✓ Validate text using Regular Expressions
- ✓ Use Iterators, Generators, Lambda, map(), filter(), reduce()
- ✓ Apply Decorators, Context Managers & Scope rules
- ✓ Debug tracebacks, write test cases & follow PEP 8
- ✓ Build complete file-based modular Python applications
- ✓ Solve complex coding challenges independently
- ✓ Strong foundation for Analytics, Automation, AI & Backend

ASSESSMENT & CERTIFICATION

Your progress is evaluated through continuous programming assignments, logic challenges, exercises, capstone project, and viva.

Score Range	Grade
90% – 100%	A+
80% – 89%	A
70% – 79%	B+
60% – 69%	B
Below 60%	Needs Improvement

 Successful candidates receive the official **Tech NextGen Academy Core Python Programming Course Completion Certificate**.

☀️ WHY TECH NEXTGEN ACADEMY?

NOT JUST PYTHON CLASSES. A COMPLETE PROGRAMMING & APPLICATION-BUILDING EXPERIENCE.

- 🧠 **Logic-Building Focus:** Think, plan, and solve before coding.
- 🐍 **Hands-On Python:** Practical classroom coding.
- 🧩 **Problem-Solving:** Progressively challenging exercises.
- 📦 **Core Collections:** Lists, Tuples, Sets, Dictionaries.
- 🏗️ **Complete OOP:** Classes, Encapsulation, Abstraction, Inheritance, Polymorphism.
- 📁 **Modular Coding:** Functions, Modules, Packages.
- 📁 **Data Persistence:** Text, CSV, and JSON storage.
- 🔍 **Regular Expressions:** Validation & extraction.
- ⚡ **Advanced Python:** Iterators, Generators, Decorators, Context Managers.
- 🛡️ **Exception Handling:** Robust error management.
- 🔧 **Debugging & Testing:** Code quality & verification.
- 🏢 **Real-World Projects:** Management & business apps.
- 🎤 **Viva & Presentation:** Technical interview readiness.
- 📝 **Assessment Certification:** Practical evaluation.

🔥 YOUR NEXT STEP & PROGRESSION

DON'T JUST LEARN PYTHON. LEARN TO THINK, AUTOMATE & BUILD.

After completing Core Python Programming, progress toward:

Advanced Python Programming • Data Structures & Algorithms • Python Automation • Python for Data Analytics (NumPy, Pandas, Data Visualization, Power BI) • Python for AI & Machine Learning • Backend Development (Django, Flask, FastAPI, REST APIs, SQL Databases) • Full Stack Python Development • Testing & Automation • Git & GitHub • Professional Software Development

📞 JOIN TECH NEXTGEN ACADEMY

Learn Today. Lead Tomorrow.

📍 **Address:** Jadavpur, Kolkata

📞 **Phone:** +91 6292009436 / +91 6292009536

📱 **WhatsApp:** +91 6292009436

✉️ **Email:** technextgenacademy@gmail.com

🌐 **Website:** technextgenacademy.com

⚡ **Admissions:** Open | Limited Seats

START CODING. MASTER PYTHON. BUILD REAL APPLICATIONS.