



CREATOR • ADVANCED PYTHON & SYSTEM BUILDING

Build Systems.

Think Like an Engineer.

6 Months • Weekly 60-Minute LIVE Class • Small Skill-Based Cohorts

Designed for confident Python students ready to architect larger applications, integrate services, automate workflows and prepare for real AI development.

100% LIVE

CODING GYM

PROJECT-BASED

DEVELOPER MINDSET

Strong Python skills are essential to building with AI — not just using it.

Advanced Python → software systems → APIs & automation → AI application development.



PROGRAM OVERVIEW

Six months. From Builder to confident Creator.

24 LIVE CLASSES

Approximately 24 weekly sessions
60 minutes each
Hands-on coding every week
Real-time instructor guidance

CORE PYTHON

OOP → architecture → APIs → testing →
automation → data workflows →
production-minded applications

Students learn to connect components
into systems.

LEARN BY BUILDING

Advanced Coding Gym • Debugging
Gym • APIs • automation • architecture
• testing • capstone

Every project emphasizes design
decisions and maintainability.

OUTCOME

Students finish able to architect, test, debug, document and present substantial Python systems — ready to transition into AI application development.



MONTHS 1-2

ADVANCED PYTHON + SOFTWARE ARCHITECTURE

01 • ADVANCED OOP + ARCHITECTURE

classes • composition • inheritance concepts • interfaces
• separation of concerns • reusable components

Coding Gym: refactor working code into cleaner architecture

02 • ALGORITHMS + DATA MODELING

complex collections • transformations •
searching/sorting • complexity intuition • validation •
data models

Build: systems that process and transform meaningful datasets

GOAL Students move from writing programs to designing software components with clear responsibilities and intentional structure.



MONTHS 3–4

APIs + TESTING + ADVANCED DEBUGGING

03 • APIs + INTEGRATIONS

REST concepts • requests • JSON • authentication concepts • pagination • failures • rate-limit awareness

Build: applications that combine multiple external data sources

04 • TESTING + RELIABLE SOFTWARE

unit-testing mindset • assertions • edge cases • logging • exceptions • configuration • modular testability

Build: applications designed to fail safely and be easier to maintain



DEBUGGING GYM

Reproduce → isolate → inspect state → trace across modules → use logs/tracebacks → fix root cause → prevent regression

AI challenge: audit generated code for logic, security and reliability issues.

GOAL Integrate services and build reliable software that can be tested and debugged systematically across multiple components.



MONTHS 5–6

AUTOMATION + SYSTEMS + CREATOR CAPSTONE

05 • AUTOMATION + WORKFLOWS

multi-step automation • file/data pipelines • scheduled-workflow concepts • configuration • reusable services

Build tools that automate a meaningful end-to-end process.

06 • SYSTEM DESIGN + CREATOR CAPSTONE

requirements • architecture diagram • modules • APIs/data • testing • GitHub • documentation • Creator capstone

Define → architect → build → test → debug → document → demo.

GOAL Deliver a substantial system whose architecture, tradeoffs and behavior the student can explain, defend and extend.



LEARNING BY BUILDING

Not just projects — Creators engineer connected Python systems.

Data Pipelines

transform • validate • persist

Integrated Apps

multiple APIs • JSON •
resilience

Workflow Automation

pipelines • configuration •
services

Software Architecture

components • OOP •
maintainability

Advanced Debugging Gym

tracing + testing + root cause

Creator Capstone

architect • build • test •
demo

GitHub becomes an engineering portfolio: meaningful commits, README documentation, architecture notes and a polished final project.



CREATOR CAPSTONE

Real systems. Creator-level ownership.

AUTOMATED PRODUCTIVITY SYSTEM

Tasks • priorities • persistent data • external integrations • automation • reporting

Designed as a multi-module application.

MULTI-API INTELLIGENCE

DASHBOARD

Combine multiple APIs • normalize data • filter/analyze results • caching concepts • graceful failures

Designed to demonstrate system integration.

MY OWN IDEA

Student proposes a real system, defines scope and architecture with Ankit, builds a Creator-level version, tests it and documents it on GitHub.

The capstone proves the student can own a system end-to-end: requirements, architecture, implementation, testing, debugging, documentation and demo.



WHAT COMES NEXT

Creator is where software development becomes system building — and the bridge to AI engineering.

1 • EXPLORER

Foundation checkpoint
Python fundamentals
Coding + Debugging Gym
Explorer-level projects

2 • BUILDER

Core Python depth
Files + OOP basics
APIs + automation
Builder capstone

3 • CREATOR

Advanced Python systems
Architecture + testing
Automation + integrations
Creator portfolio

Recommended placement: students should demonstrate Builder-level Python and software fundamentals before Creator. Experienced students can take a readiness assessment for direct placement.

Know where your child stands in the age of AI → thinkpythonai.com/will-ai-take-my-kids-job



THINKPYTHONAI YOUNG CODERS

From “I can build software”
to “I can engineer systems.”

CREATOR • 6 MONTHS • LIVE • SMALL SKILL-BASED COHORTS

Architecture • Advanced Coding Gym • Debugging Gym • APIs • testing
• automation • GitHub • Creator capstone

Young Coders Coordinator: Crystal
+1 (650) 444-3020

THE CREATOR PROMISE

We move beyond building individual applications.

Students learn to architect connected systems.

They integrate APIs, automate workflows, test deliberately and debug across components.

Then they are ready to build AI applications on top of a real software foundation.

Strong Python skills are essential to building with AI — not just using it.