OpenEDG™ Python Institute
Teaching Python Skills for Real Careers
https://pythoninstitute.org/pycrafters

PyCrafters™ – Python for Teens

Course Overview

PyCrafters™ – Python for Teens is an engaging, hands-on introduction to coding designed for middle school students ages 11–14 (**grades 6–9**). Developed by the OpenEDG Python Institute, the course is divided into four modules spread over **~20 weeks** (about **90 hours** total). Students start with the basics of computers, operating systems, and algorithms, then move into Python programming fundamentals, creative visual programming with Turtle, Pygame, and VPython, and finish with a collaborative final project and exam prep.

The course offers two learning levels to support different age groups and abilities:

- **Explorers** (ages 11–12): guided, step-by-step learning with playful examples and simpler projects.
- **Adventurers** (ages 13–14): more independent and challenging tasks that extend concepts and encourage creative problem-solving.

Each module ends with **a mini project** where students apply what they've learned, and the course concludes with **a final team project** that integrates coding, problem-solving, documentation, teamwork, and presentation skills. Along the way, students also build awareness of workplace applications, project management tools like Kanban boards, and communication strategies that prepare them for both school and future careers.

By the end of the course, learners will be able to design, write, and debug Python programs, work with data and files, build interactive games and graphics, and present their projects with confidence.

Course Syllabus

Module 1: Introduction to Computers and Programming

Duration: ~12 hours (Weeks 1–3)

Goal: Build a foundation in computing by exploring hardware, operating systems, file management, software applications, algorithms, and pseudocode. Develop basic problem-solving strategies with algorithms and pseudocode, practice organizing projects, and prepare for coding through teamwork and communication.

Lesson 1: What Is a Computer? (2 hrs)

Objective: Understand what a computer does, the role of operating systems, and how to organize files and applications.

Topics:

- Hardware vs software
- The role of the operating system (Windows, macOS, Linux, Android, and iOS)
- Identifying common software apps: word processor, spreadsheet, presentation, and coding IDE
- Files, folders, and extensions (.txt, .docx, .xlsx, .png, .py, etc.)
- Basic file operations (create, rename, save, delete)
- Keyboarding and shortcuts (copy/paste, open, save, etc.)
- Everyday uses of computing in school/work

Explorer Activities:

- Draw a “computer as a house” analogy: hardware = walls, OS = manager, apps = furniture
- Match hardware parts to their role (CPU = brain, storage = memory)
- Explore folders: use GUI to create SchoolProject folder and save a text file
- Identify file icons and guess their type (.docx, .png, .mp3)
- Practice typing a short paragraph and saving it
- Save a Word/Google Doc and a Python file in a project folder

Adventurer Activities:

- Research: Compare two operating systems (Windows vs Linux) and share differences and similarities
- Use CLI to create and list folders (`mkdir`, `ls`/`dir`)

- Create a folder tree map for a project (School > Science > Project1)
- Rename, move, and delete files using OS tools
- Reflection: Why different OS exist and when each is useful

Lesson 2: Exploring Software Applications & Data (2 hrs)

Objective: Use software applications (spreadsheets, text editors, IDEs) to process, organize, and represent information.

Topics:
- Working with word processors, spreadsheets, presentations, and coding IDEs
- Sorting, filtering, and searching in spreadsheets
- Why spreadsheets and databases matter in the workplace (finance, HR, inventory, scheduling)
- Displaying information with charts/graphs
- How this relates to programming (programming can automate tasks)

Explorer Activities:
- Create a class schedule in a spreadsheet
- Enter class attendance into a spreadsheet and filter absent students.
- Sort list of classmates alphabetically
- Filter a list of favorite movies by genre column
- Make a simple bar chart of daily screen time

Adventurer Activities:
- Build an expense tracker in a spreadsheet (add, sort, sum)
- Generate a graph of monthly spending from data
- Compare spreadsheet vs coding → when is each better?
- Save spreadsheet as `.xlsx` and `.csv`, open both, compare differences
- Create a spreadsheet to calculate total weekly work hours.

Lesson 3: Thinking Like a Programmer: Algorithms & Problem-Solving (4 hrs)

Objective: Learn how to break down problems into algorithms and represent them with pseudocode and flowcharts.

Topics:
- What is an algorithm? Provide everyday examples
- Multiple ways to solve the same problem
- Flowcharts and pseudocode basics

- Step-by-step problem decomposition
- Links between real-world tasks and program logic

Explorer Activities:
- Pseudocode: Write steps for brushing teeth
- Flowchart: making a sandwich/cereal
- Group exercise: Create two different pseudocode solutions for "cross the street safely"
- Compare two versions – both work, but which is clearer?
- Practice keyboarding while typing pseudocode into a text editor

Adventurer Activities:
- Write pseudocode for a vending machine or traffic light system
- Design a flowchart for a simple maze or decision tree
- Translate flowchart into step-by-step instructions (not yet Python)
- Research a famous algorithm (e.g., sorting, LIFO, FIFO, etc.) and explain simply
- Debate: Why can multiple algorithms solve the same problem?

Lesson 4: Programming Languages & Careers (2 hrs)

Objective: Explore programming languages, why they differ, and how coding skills connect to real-world jobs.

Topics:
- Why many programming languages exist (design approaches, strengths)
- Python vs others (JavaScript, Swift, C++)
- Programming in careers: games, business, AI, web dev
- Teamwork and communication in coding
- How coding supports everyday workplace tasks (automation, reporting, web tools)

Explorer Activities:
- Match careers to common workplace applications
- Watch short demo of Python vs Scratch vs HTML
- Career match-up game: "Which job uses which language?"
- Write a reflection: "Where could I use coding in my life?"

Adventurer Activities:
- Research project: pick one language (Java, C#, HTML) and share its main use
- Group brainstorm: different file formats & why they exist
- Case study: How Python is used in games or AI
- Case study: How a company uses Python scripts to process data from spreadsheets

- Collaborative exercise: Write pseudocode as a team for a short problem, then code it
- Present findings in a short team presentation

Lesson 5: Mini Project – Computing Foundations (2 hrs)

Objective: Apply knowledge of computers, operating systems, software applications, algorithms, and programming languages to design a small project. Organize files, write pseudocode/flowcharts, and present the project in a short team presentation.

Topics:
- Creating and organizing project folders
- Using software applications to support project development
- Writing pseudocode and flowcharts for a chosen scenario
- Exploring multiple algorithms for the same problem
- Presenting project work clearly and concisely

Explorer Activities:
- Provide a simple checklist (paper or digital) to track project steps
- Create a Project Folder with subfolders (Docs, Data, Code, Images). Save at least one file in each (Word doc, spreadsheet, image, empty Python file)
- Work in pairs to design a flowchart of a real-life process (morning routine, crossing a street, lunch line)
- Write pseudocode for the same process, compare with another team, and discuss differences
- Create a 2-4 slide presentation describing the project (flowchart + pseudocode) and present to the class in 2 minutes

Adventurer Activities:
- Introduce a shared doc or sheet as a basic task tracker
- Choose a mini-scenario (vending machine, traffic light, or library checkout system). Write pseudocode and create a flowchart to show how it works
- Use a spreadsheet to model part of the process (e.g., vending machine stock and prices, library checkout dates)
- Save all files in a structured project folder and name them appropriately.
- Write a short README (3–5 sentences) explaining the project idea, team roles, and why algorithms can differ
- Prepare a 2-minute slideshow presentation (3–4 slides: overview, pseudocode, flowchart, and one file example) and present to the class

Module 2: Python Fundamentals

Duration: ~42 hours (Weeks 4–12)

Goal: Learn Python basics step by step, including variables, operators, conditionals, loops, functions, strings, lists, dictionaries, files, and debugging. Practice writing small programs, solving problems with code, and building confidence through a fundamentals mini project.

Lesson 6: First Steps in Python (3 hrs)

Objective: Write and run your first simple Python program, connect algorithms to code, and understand the structure of a program. Learn how computers store, manipulate, and recall information.

Topics:

- Using Python IDE (Edube Sandbox, Thonny, IDLE, or online REPL)
- First program: `print("Hello, World!")`
- Comments (`#`)
- Variables = storage boxes
- Input/output with `print()` and `input()`
- Structure of a simple program (sequence of instructions)
- Saving a program to a file

Explorer Activities:

- Run `print("Hello, World!")`
- Change it to greet three friends
- Write a program that asks for their name and prints a welcome
- Add comments explaining each line
- Write pseudocode for “introduce yourself”, then code it
- Debug: intentionally misspell `print` and read the error
- Run a .py file.

Adventurer Activities:

- Write pseudocode first, then implement in Python
- Save answers in variables, print formatted output
- Create a three-question survey using `input()`
- Group project: one student writes pseudocode, another codes it
- Debug: fix a broken `"Hello, World!"` program

Lesson 7: Numbers and Text in Python (4 hrs)

Objective: Extend your first program by performing calculations and combining text with numbers. Learn how to use operators, convert between data types, and debug type-related errors.

Topics:

- Data types: int, float, str, bool
- Type conversion (`int()`, `float()`, `str()`)
- Arithmetic operators (`+`, `-`, `*`, `/`, `//`, `%`, `**`)
- Operator precedence (PEMDAS)
- Combining strings and numbers in output
- Common type errors and debugging

Explorer Activities:

- Create a birthday program: ask for age, print "Next year you will be …"
- Ask for two numbers, print their sum and product
- Make a "Name Repeater": ask for a name, print it 5 times
- Ask for pet's name, print a sentence with it
- Debug: try `int("hello")` and read the error

Adventurer Activities:

- Tip Calculator: ask for bill + % tip → print total and split between friends
- Area Calculator: input rectangle sides → print area and perimeter
- Mini Math Quiz: generate random numbers and ask user to solve addition
- Debug challenge: fix teacher-provided program with wrong type conversions

Lesson 8: Making Choices with If Statements (6 hrs)

Objective: Teach your program to make decisions using conditions. Practice comparison operators and branching logic to handle different outcomes.

Topics:

- Boolean values (True, False)
- Comparison operators (`==`, `<`, `>`, `>=`, `<=`, `!=`)
- `if`, `elif`, and `else` statements
- Nested conditions

Explorer Activities:

- Umbrella Program: ask "Is it raining?" → suggest umbrella or sunglasses

- Pass/Fail Quiz: input score, check if ≥ 50
- Even/Odd Checker: input a number, print result
- Compare two numbers, print the larger.
- Favorite Color: input favorite color, print a custom response

Adventurer Activities:

- Rock–Paper–Scissors: two players input choice, program decides winner
- Leap Year Checker: input year → check if leap year
- Discount Calculator: if purchase > 100 → apply discount
- Quiz Grader: ask three questions, tally score and print result
- Debug nested if errors.

Lesson 9: Repetition with Loops (6 hrs)

Objective: Learn to repeat actions automatically with loops. Understand for and while loops, and how to control them.

Topics:

- `for` loops with `range()`
- `while` loops (repeat until condition is met)
- Loop control: `break`, `continue`
- Infinite loop errors and debugging

Explorer Activities:

- Print numbers 1–20
- Multiplication table (1–5)
- Countdown from 10 to 1
- Ask user to type “stop” → loop continues until correct word
- Print all even numbers up to 20

Adventurer Activities:

- Number Guessing Game: random number between 1–10 until guessed
- ASCII Art: draw a triangle with `*` using loops
- Password Retry System: allow three attempts, then block access
- Prime Number Checker (input → decide prime or not)
- Add timer delay using `time.sleep`.
- Debug challenge: fix an infinite loop bug

Lesson 10: Reusable Code – Functions and Imports (5 hrs)

Objective: Write reusable code blocks with functions and learn how to organize programs into smaller parts. Explore how to pass inputs and return outputs for modular programming, and get introduced to importing and using Python's built-in libraries (e.g., `math`, `random`).

Topics:

- Defining a function with `def`
- Calling a function
- Parameters and arguments
- Scope
- Return values with `return`
- Built-in functions vs custom functions
- Importing libraries and using library functions (`import math`, `import random`, `import time`, and `import this`)
- Sample functions: `math.sqrt()`, `math.pi`, `random.randint()`, `random.choice()`, and `time.sleep()`

Explorer Activities:

- Greeting Function: `def greet(name)` and prints `"Hello, name!"`
- Square Function: input a number → return squared
- Function for rectangle area calculation
- Reuse: call same function multiple times with different inputs
- Import `math` and use `math.sqrt()` to find the square root of user input
- Import `random` and use `random.randint(1, 6)` to simulate a dice roll

Adventurer Activities:

- Function Library: max of three numbers, dice roller
- Even/Odd Function: return `True`/`False`
- Reverse String Function: return reversed version
- Modular Calculator: separate functions for add, subtract, multiply, divide
- Use `math.pi` to calculate the area of a circle given its radius
- Program that uses `random.choice()` to pick a random item from a list
- Debug challenge: fix teacher-provided functions with errors (missing `return`, wrong indentation, etc.)
- Import `this` and explore *The Zen of Python* – discuss its simple "rules" (e.g., Simple is better than complex) and what they mean in real programming

Lesson 11: Working with Strings (3 hrs)

Objective: Explore text data in detail. Learn how to access characters, slice substrings, and apply built-in string methods to transform and analyze text.

Topics:

- Strings as sequences of characters
- Indexing (positive and negative indexes)
- Slicing (`word[0:3]`, `word[-2:]`)
- Common string methods: `.upper()`, `.lower()`, `.replace()`, `.find()`, `.count()`
- Concatenation and repetition
- Measuring string length with `len()`
- Iterating through strings with `for`

Explorer Activities:

- Input a name, print the first and last letter
- Slice a word to show the first three characters
- Count the number of characters with `len()`
- Change a sentence to uppercase and lowercase
- Replace one word in a sentence with another
- Loop through a name, printing each letter on a new line
- Debug: try accessing an out-of-range index (e.g., `word[100]`) and read the error

Adventurer Activities:

- Palindrome Checker: input a word → check if it reads the same forwards/backwards
- Word Scrambler: slice and rearrange letters in creative ways
- Search Program: ask for a word in a sentence, check if it exists using `.find()`
- Frequency Counter: count how many times a chosen letter appears
- Mini Project: input a sentence → print number of words, number of letters, and the longest word

Lesson 12: Working with Lists (3 hrs)

Objective: Store and manage multiple items using lists. Learn to add, remove, sort, and loop through lists.

Topics:

- Creating lists with `[]`
- Adding/removing items (`append`, `remove`)
- Sorting and reversing lists

- Iterating through a list with `for`

Explorer Activities:
- Create a list of favorite foods, print each one
- Add/remove an item from the list
- Sort a list of names alphabetically
- Print the number of items with `len()`

Adventurer Activities:
- Average Grade Calculator: input grades, store in list, compute average
- Weekly Expenses Tracker: input expenses, sum them up
- Search for an item in a list (user input)
- Shuffle list items with `random.shuffle`

Lesson 13: Storing Information with Dictionaries (4 hrs)

Objective: Use key–value pairs to store related information. Learn to add, update, and retrieve values from a dictionary.

Topics:
- Creating dictionaries with `{}`
- Accessing values by key
- Updating or adding new key–value pairs
- Iterating through dictionary items

Explorer Activities:
- Mini Phonebook (friend → phone number)
- Dictionary of 3 countries → capitals
- Update dictionary with a new favorite color
- Print all keys and values
- Lookup user input key

Adventurer Activities:
- Word Counter: count how many times each word appears in a sentence
- Mini Diary: store a date → note in dictionary
- Student Grades Dictionary: store grades, calculate average
- Inventory Tracker: item → quantity, update stock
- Merge two dictionaries
- Debug KeyError with `.get()`

Lesson 14: Saving and Loading Data (2 hrs)

Objective: Learn how to save program output to a file and read it back later. Understand the role of file types in data storage.

Topics:

- File types: .txt vs .csv
- Opening files for writing and reading (with `open()`)
- Writing text to files
- Reading file contents

Explorer Activities:

- Save “About Me” info (name, age, hobby) to a text file
- Write a list of favorite foods → save to file
- Open file and print its contents
- Debug: try opening a non-existent file

Adventurer Activities:

- Expense Tracker: save expenses to a file, reload on next run
- Simple Log File: append new entries every run
- Read CSV-style text file and calculate average
- Safe file open with error handling

Lesson 15: Debugging Basics (2 hrs)

Objective: Learn to identify, understand, and fix common coding mistakes. Practice using error messages and exception handling.

Topics:

- Types of errors: syntax, runtime, logic
- Reading error messages
- Debugging systematically (print functions, tracing values)
- Handling runtime errors with simple `try`/`except` mechanisms

Explorer Activities:

- Fix three short buggy programs (teacher-provided)
- Handle divide-by-zero gracefully
- Catch input type error with try/except
- Print variable values to trace bugs

Adventurer Activities:

- Multi-error handling: catch both `ValueError` and `ZeroDivisionError`
- Debug nested conditionals (fix incorrect logic)
- Robust Calculator: add error handling for bad inputs
- Peer Debugging Challenge: swap broken code with a classmate and fix
- Create a robust login system with error catching.

Lesson 16: Mini Project – Python Fundamentals (4 hrs)

Objective: Apply the skills from Module 2 to design and build a small project that combines variables, decisions, loops, functions, strings, collections, file I/O, and debugging. Practice teamwork, planning, and presenting a working program.

Topics:

- Choosing a project idea (game, utility, or creative app)
- Breaking project into smaller tasks (pseudocode/flowchart)
- Basic project management: planning tasks, tracking progress, and assigning roles
- Using variables, loops, conditionals, and functions together
- Saving data to files and reading it back
- Debugging and testing systematically
- Presenting project results clearly

Explorer Activities:

- Choose one project from a teacher-provided menu (examples below):
    - Personal Diary: user adds notes (saved to file), view past notes
    - Quiz Game: five questions with score counter
    - Expense Tracker: enter expenses, save to file, show total
    - Secret Messages: use string slicing/replacing to “encode” and “decode”
- Work in pairs: one writes pseudocode, the other codes
- Add comments to explain program steps
- Use a printed or online Kanban board or sticky notes (“To Do, Doing, Done”)
- Present project to classmates in 2-3 minutes (explain what it does)

Adventurer Activities:

- Brainstorm own project idea (teacher approval), or choose one of the below:
    - Multi-level Quiz with scoring and file-based leaderboard
    - Student Grades Manager: input grades → calculate average, save to file, reload
    - Inventory Tracker: store items in a dictionary, update quantities, save/load
    - Text Analyzer: count words, letters, longest word, save report to file
- Plan project with pseudocode and flowchart

- Use functions to separate different features (input, calculation, file save/load)
- Debug program together as a team (peer code review)
- Write a short README (2-3 lines: what it does, how to use it)
- Present project to class (5 minutes: features + one bug fixed during coding)

Module 3: Visual & Interactive Programming

Duration: ~28 hours (Weeks 13–18)

Goal: Use Python's beginner-friendly libraries (Turtle, Pygame, VPython) to create visual programs, drawings, animations, and simple games. Apply programming concepts in creative, interactive projects and strengthen debugging and problem-solving through hands-on practice.

Lesson 17: Turtle Graphics – Drawing Basics (4 hrs)

Objective: Use the Turtle library to draw simple shapes and explore digital art. Learn about coordinates, colors, and movement commands.

Topics:
- Importing and aliasing
- Importing Turtle (`import turtle as t`)
- Moving the turtle (forward, backward, left, right)
- Drawing shapes (line, square, triangle, circle)
- Setting colors and pen size
- Coordinates and positioning

Explorer Activities:
- Draw a line of 5 circles in a row
- Draw a triangle, square, and pentagon
- Change pen color and size
- Move turtle to a new spot with penup and pendown
- Write text with Turtle
- Draw a smiley face using circles

Adventurer Activities:
- Draw a house with a square and triangle roof
- Add a sun with rays in the corner
- Combine shapes into a traffic light (three circles stacked)
- Create a flower using a loop

- Debug: fix a drawing where shapes don't close properly

Lesson 18: Turtle Graphics – Patterns & Functions (4 hrs)

Objective: Create complex drawings by reusing code and loops. Learn to use functions in Turtle to build modular art.

Topics:
- Using `for` loops to repeat shapes
- Functions for modular drawings
- Artistic effects: fill colors, pen size, penup/pendown
- Creating repeating patterns

Explorer Activities:
- Draw five stars in a row
- Create a hexagon flower pattern
- Experiment with pen sizes and colors
- Fill a shape with color
- Reuse a function to draw multiple stars

Adventurer Activities:
- Create a mandala or spirograph with loops
- Build a colorful snowflake pattern
- Recursive fractal tree (guided template provided)
- Draw a simple maze with lines
- Challenge: final digital artwork with functions

Lesson 19: Intro to Pygame – Drawing & Windows (4 hrs)

Objective: Explore 2D graphics with Pygame by creating windows, shapes, and text. Learn about refresh loops.

Topics:
- Setting up a Pygame window
- Drawing shapes (circle, rectangle, polygon)
- Choosing colors and background
- Updating the display in a loop
- Handling the quit event

Explorer Activities:

- Open a Pygame window with set dimensions
- Draw a red circle in the center
- Fill the background with blue
- Draw a rectangle and triangle
- Add text to the window

Adventurer Activities:
- Random circles each time program runs
- Move a circle slowly across screen (auto-animation)
- Draw a checkerboard pattern
- Program that changes background color every frame
- Debug: fix a program that crashes immediately

Lesson 20: Pygame – Adding Movement (4 hrs)

Objective: Make graphics interactive using keyboard and mouse input. Add sprite movement and simple sounds.

Topics:
- Event handling in Pygame
- Keyboard input for movement
- Mouse-click events
- Adding sounds
- Keeping objects inside screen boundaries

Explorer Activities:
- Move a square with arrow keys
- Change square color while moving
- Add a boundary (stop at window edges)
- Play a sound when a key is pressed
- Draw a circle on mouse click

Adventurer Activities:
- Ball bouncing with paddle
- Sprite that follows mouse pointer
- Add background image
- Add sound when objects collide
- Debug: fix input lag or “stuck” key press

Lesson 21: Pygame – Building a Simple Game (4 hrs)

Objective: Combine movement, collisions, and scoring to make a basic game. Practice game loops and logic.

Topics:
- Collision detection
- Game loop structure
- Score counters and lives
- Game over and restart conditions

Explorer Activities:
- Falling object drops from top
- Player moves paddle to catch it
- Add a score counter
- Display "Game Over" message
- Restart game with key press

Adventurer Activities:
- Multiple falling objects with different speeds
- Add lives counter (lose when missed 3 times)
- Level system: increase speed after score milestones
- Show leaderboard at end of game
- Debug collision detection issues

Lesson 22: VPython – 3D Graphics (4 hrs)

Objective: Visualize and animate 3D objects using VPython. Learn about shapes, animations, and scientific models.

Topics:
- Setting up VPython
- Creating 3D objects (sphere, box, cylinder)
- Positioning objects in 3D space
- Animating motion with loops
- Building simple models (solar system, pendulum)

Explorer Activities:
- Display a 3D sphere
- Add a cylinder as a pillar

- Animate a bouncing ball
- Rotate a cube in place
- Change background color

Adventurer Activities:
- Animate Earth orbiting the Sun
- Add Moon orbiting Earth
- Bouncing ball with gravity effect
- Pendulum swing simulation
- Debug: fix object that moves in wrong direction

Lesson 23: Mini Project – Visual Creation (4 hrs)

Objective: Apply Turtle, Pygame, or VPython to design a small creative project. Practice planning, coding, debugging, and presenting a finished product.

Topics:
- Choosing a project type (art, animation, game, or simulation)
- Breaking down project into smaller steps
- Using functions and loops to organize code
- Debugging visual programs
- Presenting project to peers

Explorer Activities:
- Turtle Project: draw a digital postcard (house, tree, sun)
- Turtle Art: create a repeating pattern or mandala
- Pygame Project: simple catching game (falling objects, paddle)
- VPython Project: bouncing ball animation
- Present project to a partner: explain what it does and how it works

Adventurer Activities:
- Turtle Project: draw a city skyline or flower garden with loops
- Pygame Project: game with scoring and levels (catching objects, avoid falling ones)
- VPython Project: mini solar system with Earth + Moon orbit
- Add extra features: sound effects, multiple objects, or scoreboard
- Present project to class: explain concept, challenges, and how you debugged it

Module 4: Final Project & Exam Prep

Duration: ~8 hours (Weeks 19-20)

Goal: Plan, build, and present a final project that demonstrates coding skills, teamwork, communication, and documentation. Review and practice key course concepts while preparing for the certification-style exam.

Lesson 24: Final Project Development (4 hrs)

Objective: Plan, design, and begin building a final project that demonstrates coding knowledge, teamwork, and communication skills.

Topics:

- Brainstorming project ideas (game, simulation, art, or utility app)
- Writing pseudocode and flowcharts for planning
- Using simple project management tools (task lists, shared docs, Kanban boards)
- Dividing roles in a team (designer, coder, tester, presenter)
- Writing clear documentation and comments

Explorer Activities:

- Brainstorm project ideas in small groups (teacher provides project menu: game, art, diary, quiz, etc.)
- Write pseudocode for chosen project
- Set up a digital Kanban board to plan tasks and update progress
- Create a task list for project steps (who does what)
- Begin coding one feature as a team
- Document code with simple comments
- Share progress with class using short oral update

Adventurer Activities:

- Choose and define a bigger project (multi-level game, multi-function tool, or 3D simulation)
- Use a simple project management tool (Google Docs or whiteboard Kanban)
- Write pseudocode and flowchart collaboratively for core features
- Assign team roles (developer, tester, documentation writer)
- Begin building core modules with functions and loops
- Write a README file explaining project purpose, usage, and team members

Lesson 25: Exam Prep & Presentations (3 hrs)

Objective: Review course concepts, practice debugging, and present final projects with clear communication and reflection.

Topics:

- Reviewing variables, conditionals, loops, functions, lists, dicts, strings, and files
- Debugging practice with broken programs
- Preparing presentations (oral + visual)
- Peer evaluation and feedback
- Online communication etiquette (sharing code safely, commenting positively)

Explorer Activities:

- Review quiz on Python basics (10 quick problems)
- Debug three short buggy programs
- Prepare a 2-minute demo of final project
- Share what they learned in a reflection paragraph
- Peer feedback: one positive comment, one suggestion
- Practice Test

Adventurer Activities:

- Advanced review quiz (code snippets with bugs or missing pieces)
- Debug a broken Pygame or Turtle program
- Prepare a 5-minute presentation explaining project design, challenges, and solutions
- Post code + documentation to a shared online space (Google Drive, GitHub Classroom, or LMS)
- Write a reflection report: what went well, what they'd improve

Lesson 26: PyCrafters™ Final Exam (45 min)

Objective: Demonstrate mastery of the course content by completing a proctored, certification-style exam.

Exam Details:

- Duration: 45 minutes
- Format: 32 exam items (single-select, multiple-select, interactive & short coding tasks)
- Content coverage: all major topics from Modules 1–4 (computers & OS, Python fundamentals, visual/interactive programming, problem-solving, debugging, and teamwork)
- Achievement levels: Explorer Level – pass with a score between 70–85%, Adventurer Level – pass with a score between 86–100%.

Students who successfully pass the exam will receive **a digital certificate and badge via Credly**, recognizing their achievement in Python fundamentals and creative coding.

Course Summary

The table below shows how the course is structured, with each module's suggested hours, main topics, and learning goals.

Module	Hours	Weeks	Lessons	Main Topics	Learning Goals
Module 1: Introduction to Computers and Programming	~12 hrs	1-3	1-5	Computers & operating systems, files & folders, software apps, spreadsheets, algorithms, pseudocode, programming languages, mini project	Learn how computers and operating systems work, organize files, use common software, and start thinking like a programmer by writing pseudocode and flowcharts.
Module 2: Python Fundamentals	~42 hrs	4-12	6-16	Python basics (variables, data types, operators), decisions, loops, functions, imports, strings, lists, dictionaries, files, debugging, mini project	Write and run Python programs, use variables and logic to solve problems, work with text and data, save files, fix errors, and practice building small programs.
Module 3: Visual & Interactive Programming	~28 hrs	13-18	17-23	Turtle graphics, patterns & functions, Pygame (2D graphics & games), VPython (3D graphics), mini project	Create visual programs, drawings, animations, and games. Apply coding concepts in creative projects, debug programs, and share your work with others.
Module 4: Final Project & Exam Prep	~8 hrs	19-20	24-26	Team project design & coding, project management with Kanban boards, documentation, presentations, review & exam prep, Final Exam	Work in a team to plan, build, and present a final project. Practice teamwork, communication, and project management skills while reviewing Python fundamentals.
	~90 hrs	**20**	**26**		