Intro to Python — CS++ Syllabus

Provider: Yellow Dart Publishing (DBA of Kevin P Hare LLC) **Course platform:** CS++ (csplusplus.com) **Syllabus version:** v1.0 (2026-06-10) **Course type:** Free, beginner-friendly introductory programming course (not an AP course) **Intended use:** Share with your district, counselors, or course-approval committee, or reference in classroom planning.

Intro to Python is a complete first programming course: 10 units, 30 lessons, 75 auto-checked inline coding challenges, and an in-browser Python IDE. Students go from their first `print()` statement to a multi-function capstone application built on loops, functions, lists, and dictionaries.

Companion pages on the platform: • **Day-by-Day Curriculum** — `/intropython-curriculum` • **Course Overview** — `/intropython`

Course Description

Intro to Python is a one-semester (or stretched full-year) introductory programming course for students with **no prior coding experience**. Students learn to read, write, trace, and debug real Python programs — starting from a single `print()` statement and finishing with a menu-driven, multi-function capstone application.

The course is deliberately scoped for beginners. It covers the procedural core of Python — sequencing, variables and types, arithmetic operators, strings, booleans and conditionals, loops, functions, lists, dictionaries, and tuples — and intentionally **excludes object-oriented programming and recursion**, which belong in a second course.

Every concept is taught through runnable code. Each of the 30 lessons embeds **auto-checked inline Python challenges** (75 across the course) that execute a real Python interpreter in the browser and give students instant pass/fail feedback with actual program output. Misconceptions surface during the lesson, not on the unit test.

Audience & Prerequisites

- **Audience:** middle and high school students (grades 7–12) taking their first programming course; also suitable as a gentle on-ramp before AP Computer Science Principles or an introductory Java course.
- **Prerequisites:** none. No prior programming experience is assumed. Comfort with basic arithmetic (order of operations, remainders) is helpful for Unit 3.
- **Pathways:** graduates of this course are well prepared for **AP Computer Science Principles** (the algorithm and abstraction vocabulary transfers directly) or for **CS++ Intro to Java** (`/introjava-curriculum`) as a second language.

Materials & Platform

Everything the course needs is **free and browser-based** — no installations, no licenses, no lab software:

- **CS++ Lessons** — 30 lessons across 10 units at `/intropython/dashboard`, each with auto-checked inline Python challenges.
- **CS++ Python IDE** (`/python-ide`) — an in-browser, multi-file Python environment used for every lab. Works on Chromebooks, lab machines, and personal devices.
- **Practice banks** — quiz practice (`/intropython/quiz-practice`), a 50-question MCQ bank (`/intropython/mcq-practice`), Parsons puzzles (`/intropython/parsons`), and vocabulary flashcards (`/intropython/flashcards`).
- **Day-by-Day Curriculum** (`/intropython-curriculum`) — a classroom-ready 90/180-day pacing guide with objectives, timed procedures, materials, success criteria, differentiation strategies, and exit tickets for every day.
- **Classroom tools** — teachers can create a free classroom, invite students, assign lessons and practice banks, and track per-student progress.

Recommended (not required): paper trace-table templates and a printed syntax reference card per unit — both described in the day-by-day curriculum.

Course Skills

Four recurring skill strands organize the course. Every unit develops all four, shifting emphasis from reading single statements early on to designing complete programs by the capstone.

Strand	Name	Description
S1	Write & Run Code	Express ideas as working Python — from prompts, from starter code, and finally from a blank file. Practiced daily through inline challenges and IDE labs.
S2	Read & Trace Code	Predict program output before running it. Trace tables follow variables through conditionals, loops, and function calls.
S3	Debug & Test	Read tracebacks (SyntaxError, NameError, TypeError, ValueError, IndexError, KeyError), locate the failing line, fix it, re-run. Later units add input validation and edge-case testing.
S4	Decompose & Build	Break problems into named functions and choose the right data structure for the job. Assessed by unit labs and the capstone.

Unit-by-Unit Outline

The course totals **90 class days** on a block/semester schedule (approximately 135 contact hours at 90 minutes per day), or 180 days on a traditional full-year schedule. Each unit combines lesson days, practice days (quiz/MCQ/Parsons), at least one hands-on IDE lab, and a low-stakes unit checkpoint.

Unit	Title	Lessons	Class Days (90-day plan)
1	Getting Started with Python	3	6
2	Variables & Data Types	3	7
3	Operators & Expressions	3	6
4	Strings	3	7
5	Booleans & Conditionals	3	7
—	Mid-Course Review & Skills Check	—	3
6	Loops & Iteration	3	8
7	Functions	3	8
8	Lists	3	8
9	Dictionaries	3	7
10	Files, Tuples & a Final Project	3	6
—	Capstone Project: Inventory Tracker	—	12
—	Course Wrap & Next Steps	—	5
	Total	**30**	**90**

Unit 1: Getting Started with Python (6 days, ~9 hours)

Students write and run their first programs on day one.

Lesson	Title	Key Topics
1.1	Your First Python Program	`print()`, text in quotes (strings), comments with `#`
1.2	Printing and Comments	Multiple `print()` calls, `sep=` and `end=`, mixing strings and numbers, meaningful comments
1.3	How Python Runs + Errors	Interpreter vs compiler, reading tracebacks, fixing SyntaxError / NameError / TypeError

Also includes: Python IDE orientation day, unit practice day, unit checkpoint.

Unit 2: Variables & Data Types (7 days, ~10.5 hours)

Lesson	Title	Key Topics
2.1	Variables and Numbers	Assignment, `int` vs `float`, dynamic typing
2.2	Strings and Booleans	`str` and `bool`, checking types with `type()`, converting between types

Lesson	Title	Key Topics
2.3	Type Conversion & input()	`input()` always returns a string; `int()`, `float()`, `str()`; common conversion errors

Also includes: interactive-program lab, Parsons day, cumulative MCQ day, checkpoint.

Unit 3: Operators & Expressions (6 days, ~9 hours)

Lesson	Title	Key Topics
3.1	Arithmetic Operators	The five arithmetic operators, precedence, why `/` always produces a float
3.2	Integer Division and Modulus	`//` floor division and `%` modulus — the whole-number workhorses
3.3	abs, round, min, max & Mixing Types	Built-in math functions; result types when `int` and `float` mix

Also includes: change-maker / time-converter lab built on `//` and `%`, practice day, checkpoint.

Unit 4: Strings (7 days, ~10.5 hours)

Lesson	Title	Key Topics
4.1	Strings and f-strings	Concatenation, repetition, f-strings, `len()`
4.2	Indexing and Slicing	0-based indexing, negative indexing, slicing substrings
4.3	String Methods	`upper()`, `lower()`, `strip()`, `replace()`, `find()`, `split()`, `join()`, the `in` operator

Also includes: text clean-up lab, Parsons day, cumulative MCQ day, checkpoint.

Unit 5: Booleans & Conditionals (7 days, ~10.5 hours)

Lesson	Title	Key Topics
5.1	Comparisons and Booleans	`== != < > <= >=` and the True/False values they produce
5.2	if / elif / else	Programs that choose different paths; multi-way decisions
5.3	Logical Operators: and, or, not	Combining conditions into one expression

Also includes: branching decision-program lab, practice day, Parsons day, checkpoint.

Mid-Course Review & Skills Check (3 days, ~4.5 hours)

Cumulative review of Units 1–5, a timed cumulative skills check, and a student-choice build day in the Python IDE.

Unit 6: Loops & Iteration (8 days, ~12 hours)

Lesson	Title	Key Topics
6.1	for Loops and range()	`range()` with one, two, and three arguments; looping over strings
6.2	while Loops	Condition-driven loops, accumulators, `break` / `continue`, avoiding infinite loops
6.3	Nested Loops & Patterns	Grids, tables, and patterns; running totals across nested iteration

Also includes: number-guessing-game lab, loop-tracing clinic (trace tables), Parsons day, practice day, checkpoint.

Unit 7: Functions (8 days, ~12 hours)

Lesson	Title	Key Topics
7.1	Defining Functions	`def`, parameters, return values, functions that return `None`
7.2	Parameters & Return Values	Multiple arguments, default values, **return vs print**
7.3	Scope & Helper Functions	Local scope, composing helper functions, docstrings

Also includes: refactoring lab (turn an earlier program into clean functions), return-vs-print clinic, Parsons day, practice day, checkpoint.

Unit 8: Lists (8 days, ~12 hours)

Lesson	Title	Key Topics
8.1	Creating and Accessing Lists	List literals, indexing (incl. negative), `len()`, changing elements in place
8.2	List Methods & Looping	`append()`, `pop()`, `remove()`, the `in` operator, for-each loops, `sum` / `max` / `min` / `sorted`
8.3	List Algorithms & Comprehensions	Building and filtering lists in loops; list comprehensions

Also includes: class-stats-calculator lab, list-algorithms clinic (accumulate / filter / find-max), Parsons+MCQ day, practice day, checkpoint.

Unit 9: Dictionaries (7 days, ~10.5 hours)

Lesson	Title	Key Topics
9.1	Creating and Using Dictionaries	Dict literals, square-bracket access, `in` for keys, `len()`
9.2	Dictionary Methods & Looping	`keys()`, `values()`, `items()`, looping over dicts, safe lookup with `get()`
9.3	Counting & Grouping with Dicts	Frequency counting, lookup tables, grouping by category

Also includes: word/character frequency-counter lab, cumulative MCQ day, practice day, checkpoint.

Unit 10: Files, Tuples & a Final Project (6 days, ~9 hours)

Lesson	Title	Key Topics
10.1	Tuples and Multiple Return	Tuples, unpacking, returning multiple values from a function
10.2	Putting It Together	One integrated program combining functions, lists, dicts, loops, and f-strings
10.3	Capstone Project: Inventory Tracker	Capstone spec walkthrough; mapping features to course concepts

Also includes: mixed Parsons day, program-extension lab, blank-file integration clinic, cumulative checkpoint over Units 1–10.

Capstone Project: Inventory Tracker (12 days, ~18 hours)

Students plan, build, test, and present a menu-driven, multi-function **Inventory Tracker** with a dictionary data model, input validation, and formatted reports:

1. Project spec, feature list, and plan
2. Sprint 1 — data model + menu loop skeleton
3. Sprint 2 — core features written as functions
4. Milestone demo + structured peer feedback
5. Sprint 3 — stretch features
6. Sprint 4 — input validation + edge cases
7. Testing day — test plans and bug hunt
8. Polish — docstrings, naming, comments
9. Presentation prep + two presentation days, plus reflection

Specs are offered at **three complexity tiers** so every student finds an entry point; stretch features challenge early finishers. Deliverables are flexible: live demo, recorded walkthrough, or annotated code tour.

Course Wrap & Next Steps (5 days, ~7.5 hours)

Full-course recap (flashcard marathon + concept map), mixed MCQ and Parsons/quiz practice, a pathways day previewing Intro to Java and AP CSP, and a celebration / portfolio-share day.

Assessment & Grading Suggestions

Assessment is continuous and mostly formative. A suggested grade composition:

Category	What It Includes	Suggested Weight
Inline challenges & exit tickets	75 auto-checked lesson challenges; daily exit tickets	15%
Practice sets	Quiz practice, MCQ bank, and Parsons puzzle days	10%
IDE labs	One hands-on build per unit (change-maker, guessing game, frequency counter, etc.)	20%
Unit checkpoints	End-of-unit quizzes drawn from the practice banks	25%
Mid-course skills check	Timed cumulative check covering Units 1–5	10%
Capstone project	Inventory Tracker: plan, sprints, testing, polish, presentation	20%
Total		**100%**

Notes:

- **Checkpoints are low-stakes by design** — their job is an honest signal before the next unit builds on this one. Students write corrections for every missed item.
- Teachers using CS++ classrooms can assign the MCQ and Parsons banks directly, set per-assignment completion thresholds (minimum-score requirements), and monitor progress per student.
- The capstone rubric should weight working core features, code organization (functions, naming, docstrings), input validation, and the presentation.

Pacing Options

- **90-day semester (block schedule)** — one curriculum row per ~90-minute class day. Each lesson day combines a mini-lesson, the CS++ lesson with inline challenges, and IDE time; every day includes a timed procedure that sums to a full block.
- **180-day full year (traditional schedule)** — each curriculum row spans two ~50-minute periods: concept instruction and inline challenges on day one, lab/practice work and the exit ticket on day two. The 90/180 toggle on `/intropython-curriculum` renumbers the plan automatically.

Instructional Strategies

- **Auto-checked inline challenges** — every lesson embeds runnable Python checked by a real in-browser interpreter; feedback is instant and visible to the teacher.
- **Trace tables before the IDE** — reading code is taught as deliberately as writing it; predictions on paper are confirmed by running the code.
- **Parsons puzzles** — assembling scrambled programs isolates program structure from syntax recall, lowering the entry barrier for novices.
- **Error journals** — error messages are information, not failure; students log missed items with the concept tested and the corrected reasoning.
- **Pair programming** — labs use driver/navigator roles with timed swaps; the capstone adds milestone demos with structured peer feedback.

Academic Integrity

Students may discuss approaches and help each other debug, but submitted code must be each student's own typing and own understanding — the standing test is "can you explain every line?" Copying solutions from a classmate, a website, or an AI tool and presenting them as one's own work is an integrity violation. Teachers are encouraged to set an explicit classroom policy for AI assistants: treat them like a classmate — fine to ask "why doesn't this work?", not fine to ask for a finished solution. Unit checkpoints and the mid-course skills check are individual, closed-resource assessments.

Accessibility

The platform is fully browser-based with no installation, works on Chromebooks and shared lab machines, and supports keyboard navigation. Lessons pair every concept with both prose and runnable code. The day-by-day curriculum includes per-day differentiation and UDL strategies: printed syntax reference cards, two-tier lab scaffolds (starter code vs blank file), sentence frames for error analysis, flexible groupings, and extended-time guidance for checkpoints. All lessons and practice banks are self-paced and free, so pacing can be adjusted per student.

Contact

Questions about the course, the platform, or classroom setup: **contact@csplusplus.com**

Teachers can create a free classroom, invite students, and assign lessons and practice banks from the CS++ teacher dashboard.