Programming by Design

Computing, Representation, and Reasoning

Grade Level: 9th Grade — Format: Modular, Project-Based Yearlong Course

Instructor: Eric Allatta

Course Overview

This full-year course introduces computer science as a discipline of structured reasoning, communication, and system design. Students explore computing as both a technical and cultural practice: one grounded in logic and abstraction, but inseparable from authorship, ethics, and representation.

Rather than start with syntax, students begin by investigating how structured thinking allows us to build systems, interpret information, and design solutions. They move from models and metaphors to code and communication—building fluency across representations and reflecting on what it means to "think computationally."

Guiding Principles:

- Programming is a language for structured thinking
- Data is never neutral—it must be interpreted and questioned
- Systems must be tested, visualized, and explained
- Good code is communicative: to machines *and* to people
- Fluency comes from routine: structured practice in writing, naming, diagramming, and revising
- Tools are thinking tools: CLI, markdown, version control, and diagrams support student authorship and structure

Conceptual Arc

Unit	Focus	Tool/Language
0	Orientation: The Story of Data	Systems + Representation (no code)
1	Functional Design	Racket
2	Data Science + Representation	Pyret
3	Control + State	Python + EarSketch
4	Interfaces + Interpretation	HTML/CSS
5	Inquiry + Real Data	APIs + Jupyter
6	Infrastructure + Power	Internet Simulator + CLI
7	Capstone	Student Choice

Strategic Pathways

1. AP Computer Science A

- Emphasis on control structures, decomposition, and algorithmic reasoning
- Strong foundation for Java syntax and object-oriented thinking

2. Data Science + Capstone

- Literate computing using notebooks and real-world data
- Emphasis on communication, reproducibility, and inquiry

3. Algebra 2 + Computing Integration

- Function modeling, conditional logic, and recursion
- Strong cross-disciplinary links between CS and math reasoning

Appendix: Selected Unit Summaries

Unit 0: The Story of Data —Orientation Through Structure and Representation

Framing Concept: Computing is a human system of structured communication.

This is a non-coding unit. Students begin with the intellectual and cultural foundations of computer science—exploring computation as a set of human decisions about meaning, structure, and systems. They practice thinking precisely, reflecting honestly, and diagramming systems that shape their lives.

Key Outcomes:

- Understand computing as a system of structured representation
- Build fluency with core metaphors (systems, abstraction, encoding, authorship)
- Normalize error, reflection, and identity as part of learning
- Establish durable routines: journals, diagrams, and prompt logs

Unit 1: Programming by Design (Racket)

Framing Concept: A program is a structured solution to a problem.

Students use Racket to design pure functions using the design recipe: contract, examples, definition, and tests. They build recursive and piecewise functions and apply them to visual and mathematical models.

Key Outcomes:

- Write and explain testable functions
- Practice decomposition and reuse
- Begin recursive reasoning aligned with algebraic logic

Unit 2: Data Science and Representation (Pyret)

Framing Concept: Data is a constructed lens on the world.

Students manipulate tabular datasets to explore representation and omission. They apply `filter`, `map`, and `build-column` functions to real museum or cultural datasets. Data visualizations support critical interpretation.

Key Outcomes:

- Use Boolean expressions and functions to analyze data
- Create and critique graphs and transformations
- Explore the role of categorization and metadata in shaping knowledge

Unit 3: Systems and Control (Python + EarSketch)

Framing Concept: Programs model dynamic systems through control flow and state.

Students shift into procedural thinking using loops, conditionals, and accumulators. In EarSketch, they create rule-based sound compositions; in vanilla Python, they simulate behaviors and interactive systems.

Key Outcomes:

- Write code using `for`, `while`, `if/else` constructs
- Understand mutation, state, and behavior
- Trace and debug procedural logic with intention

Capstone Preview

The course concludes with an independent capstone. Students synthesize learning through a data inquiry, system simulation, or interface project.

Deliverables:

- Final artifact (code, visualization, or site)
- Technical documentation + logic explanation
- Reflective writing and presentation