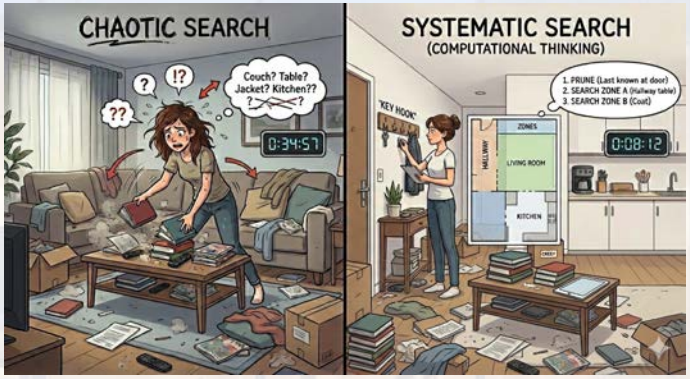
What is Computational Thinking?

(And Why It's Getting More Important With AI)

Daniel Huttenlocher
June 2026

Computational Thinking is Not Programming

- A way of framing problems and solutions so they can be solved systematically
  - Whether by a person, organization, or machine



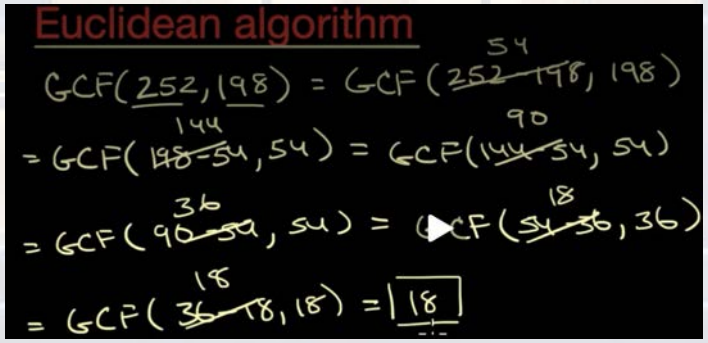
Computational Thinking is Not Programming

- A way of framing problems and solutions so they can be solved systematically
  - Whether by a person, organization, or machine
- Predates computers by millennia
  - Euclid, Renaissance-era bankers, Florence Nightingale

MIT Schwarzman College of Computing

Computational Thinking is Not Programming

- A way of framing problems and solutions so they can be solved systematically
  - Whether by a person, organization, or machine
- Predates computers by millennia
  - Euclid, Renaissance-era bankers, Florence Nightingale
- Requires posing problems in a form that can be solved computationally
  - Enabling structured solutions and verifiable results



MIT Schwarzman College of Computing

Computational Thinking is Not Programming

- A way of framing problems and solutions so they can be solved systematically
  - Whether by a person, organization, or machine
- Predates computers by millennia
  - Euclid, Renaissance-era bankers, Florence Nightingale
- Requires posing problems in a form that can be solved computationally
  - Enabling structured solutions and verifiable results
- Now, with AI coding tools, computational thinking is more critical than ever
  - Both before and after writing code
  - And relevant to use of AI beyond coding
  - Becoming more valuable, not less

Computational Thinking Maxims

- Programming is to computational thinking what writing is to clear thought
  - An incarnation of it, not the thing itself

- Most real problems arrive in a form that can't readily be solved
  - The challenge is transforming them into one that can – computationally

- Choosing the right framing is much of the work
  - The wrong one can make problems impossible; the right one can make them straightforward

Five Core Computational Thinking Practices

- *Pose the problem* – in a form that can be solved computationally
  - Something that can be approached step by step, with verifiable outcomes

MIT Schwarzman College of Computing

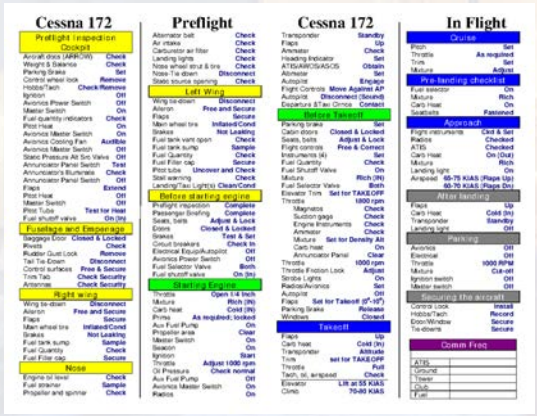
Five Core Computational Thinking Practices

- *Pose the problem* – in a form that can be solved computationally
  - Something that can be approached step by step, with verifiable outcomes
- *Abstract* – decide what to ignore
  - London Tube map: geographically wrong, good for questions riders ask

MIT Schwarzman College of Computing

Five Core Computational Thinking Practices

- *Pose the problem* – in a form that can be solved computationally
  - Something that can be approached step by step, with verifiable outcomes
- *Abstract* – decide what to ignore
  - London Tube map: geographically wrong, good for questions riders ask
- *Decompose* – structure the unmanageable into the manageable
  - Preflight checklist, recipe for baking a cake



MIT Schwarzman College of Computing

Five Core Computational Thinking Practices

- *Pose the problem* – in a form that can be solved computationally
  - Something that can be approached step by step, with verifiable outcomes

- *Abstract* – decide what to ignore
  - London Tube map: geographically wrong, good for questions riders ask

- *Decompose* – structure the unmanageable into the manageable
  - Preflight checklist, recipe for baking a cake

- *Design the procedure* – with testable steps
  - Airline recovery from a snowstorm: replanning crews, planes, passengers — quality of the procedure is the difference between a hiccup and a meltdown

Five Core Computational Thinking Practices

- *Pose the problem* – in a form that can be solved computationally
  - Something that can be approached step by step, with verifiable outcomes

- *Abstract* – decide what to ignore
  - London Tube map: geographically wrong, good for questions riders ask

- *Decompose* – structure the unmanageable into the manageable
  - Preflight checklist, recipe for baking a cake

- *Design the procedure* – with testable steps
  - Airline recovery from a snowstorm: replanning crews, planes, passengers — quality of the procedure is the difference between a hiccup and a meltdown

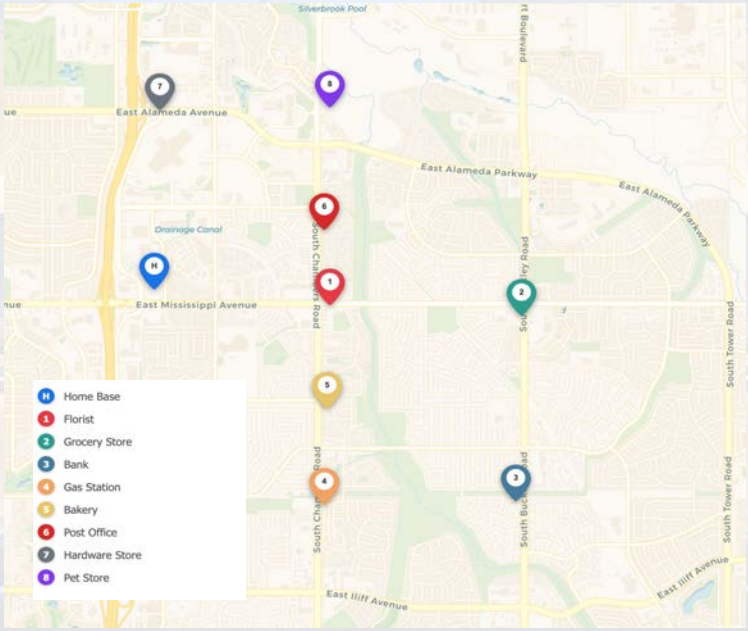
- *Verify* – how do you know the computation produced the intended result
  - Double-entry bookkeeping: every transaction recorded twice – the original error-detecting system

Computational Thinking Sketch – Saturday Errands Before Lunch

MIT Schwarzman College of Computing

Computational Thinking Sketch – Saturday Errands Before Lunch

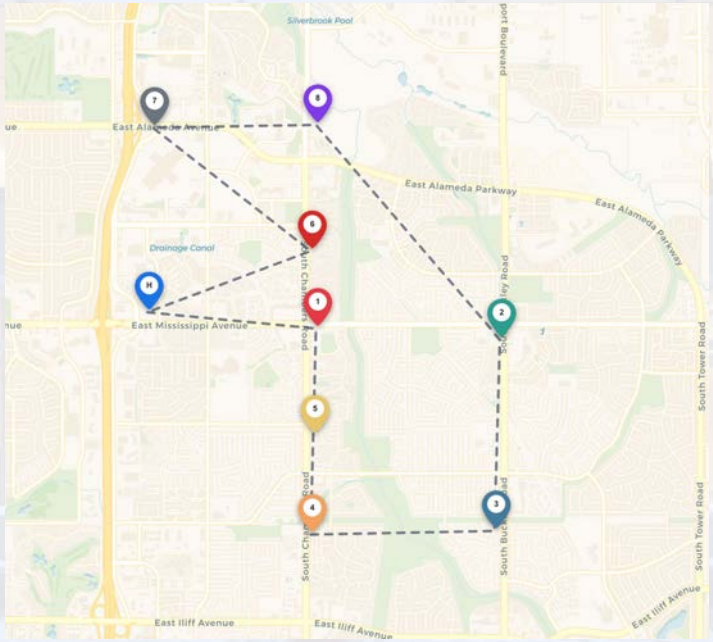
- Pose more specifically – given these stops, departing now, find a route to visit them that minimizes total driving time and returns me home for lunch



MIT Schwarzman College of Computing

Computational Thinking Sketch – Saturday Errands Before Lunch

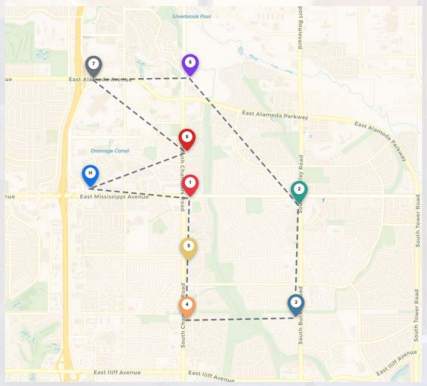
- Pose more specifically – given these stops, departing now, find a route to visit them that minimizes total driving time and returns me home for lunch
- Abstraction – model as node-edge graph with expected driving times on edges



MIT Schwarzman College of Computing

Computational Thinking Sketch – Saturday Errands Before Lunch

- Pose more specifically – given these stops, departing now, find a route to visit them that minimizes total driving time and returns me home for lunch
- Abstraction – model as node-edge graph with expected driving times on edges
- Decomposition – subproblems of (1) determining order minimizing total time, (2) finding best route from each to the next (already in map apps)



MIT Schwarzman College of Computing

Computational Thinking Sketch – Saturday Errands Before Lunch

- Pose more specifically – given these stops, departing now, find a route to visit them that minimizes total driving time and returns me home for lunch

- Abstraction – model as node-edge graph with expected driving times on edges

- Decomposition – subproblems of (1) determining order minimizing total time, (2) finding best route from each to the next (already in map apps)

- Designing the procedure – simple solution: starting from home, always drive to nearest unvisited stop, repeat – accounting for constraints, e.g., opening hours

Computational Thinking Sketch – Saturday Errands Before Lunch

- Pose more specifically – given these stops, departing now, find a route to visit them that minimizes total driving time and returns me home for lunch

- Abstraction – model as node-edge graph with expected driving times on edges

- Decomposition – subproblems of (1) determining order minimizing total time, (2) finding best route from each to the next (already in map apps)

- Designing the procedure – simple solution: starting from home, always drive to nearest unvisited stop, repeat – accounting for constraints, e.g., opening hours

- Verification / failures – what if a store is unexpectedly closed, or there is unanticipated traffic?

Computational Thinking Sketch – Saturday Errands Before Lunch

- Pose more specifically – given these stops, departing now, find a route to visit them that minimizes total driving time and returns me home for lunch

- Abstraction – model as node-edge graph with expected driving times on edges

- Decomposition – subproblems of (1) determining order minimizing total time, (2) finding best route from each to the next (already in map apps)

- Designing the procedure – simple solution: starting from home, always drive to nearest unvisited stop, repeat – accounting for constraints, e.g., opening hours

- Verification / failures – what if a store is unexpectedly closed, or there is unanticipated traffic?

Good enough for simple errands, not larger problems such as delivery routes – classical *Traveling Salesman* problem, finding best solution not feasible; scalability uses highly sophisticated approximation algorithms

MIT Schwarzman College of Computing

Classes on Computational Thinking at MIT

- 6.100 – Foundational skills in programming and computational modeling, covers programming concepts using Python, introduces algorithmic complexity and some common libraries
- 16.C11/6.C11 – Introduction to computational principles that underlie autonomous robots and vehicles, with a focus on estimation, planning, and control
- 18.C06/6.C06 – Introduction to linear algebra and optimization, for understanding complex systems and extracting structure from data (including vectors, matrices, eigenvalues, singular values, least squares, convex optimization, linear/quadratic programming, gradient descent, and regularization)
- 16.C20/18.C20 – Introduction to computational algorithms in engineering and science (natural and social) to simulate time-dependent phenomena, optimize and control systems, and quantify uncertainty in problems involving randomness
- x.Cxx – Forthcoming subject “Algorithms in Action”

AI Coding and Computational Thinking

A chatbot or coding-specific AI can address the problem of writing code, but this exposes other critical challenges in developing reliable software

- Appearing to work vs. truly doing what's intended
- Working vs. working and maintainable
- System architectural tradeoffs – e.g., efficiency, flexibility
- Security

Particularly important for code that will be used by others – these have long been bottlenecks, which are exacerbated by easier code generation with AI

AI Coding and Computational Thinking

A chatbot or coding-specific AI can address the problem of writing code, but this exposes other critical challenges in developing reliable software

- Appearing to work vs. truly doing what's intended
- Working vs. working and maintainable
- System architectural tradeoffs – e.g., efficiency, flexibility
- Security

Particularly important for code that will be used by others – these have long been bottlenecks, which are exacerbated by easier code generation with AI

Computational thinking maxims still apply, maybe more so

Computational Thinking and Agentic AI Coding

AI agents can go beyond writing code to running software development loops of coding, validating, debugging, with human judgment and goal setting

Emerging guidelines for such development loops capture computational thinking

1. **Think Before Coding** – raise questions and tradeoffs, don't assume or hide
2. **Simplicity First** – minimum code that solves the problem, if a senior engineer would say it is overcomplicated then simplify
3. **Surgical Changes** – touch only what you must to directly meet change request
4. **Goal-Driven Execution** – define strong, verifiable success criteria so agent can loop independently, weak ones require constant oversight

Computational Thinking and Agentic AI Coding

AI agents can go beyond writing code to running software development loops of coding, validating, debugging, with human judgment and goal setting

Emerging guidelines for such development loops capture computational thinking

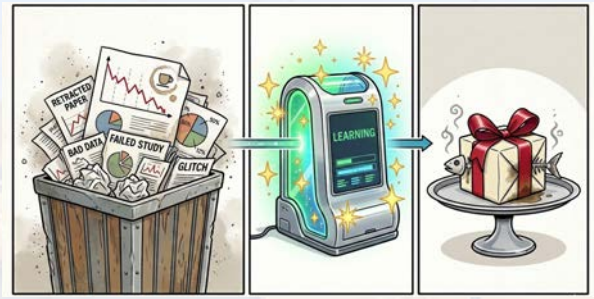
1. **Think Before Coding** – raise questions and tradeoffs, don't assume or hide
2. **Simplicity First** – minimum code that solves the problem, if a senior engineer would say it is overcomplicated then simplify
3. **Surgical Changes** – touch only what you must to directly meet change request
4. **Goal-Driven Execution** – define strong, verifiable success criteria so agent can loop independently, weak ones require constant oversight

Rapidly evolving area of practice!

MIT Schwarzman College of Computing

Why This Matters Even More Now

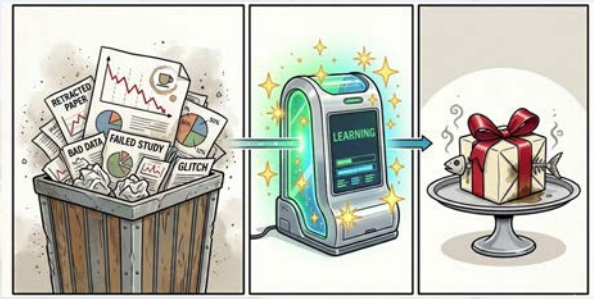
- No longer just coding – with AI a well-educated person needs to be able to pose problems computationally and use good abstractions



Or be at the mercy of systems they can't evaluate

Why This Matters Even More Now

- No longer just coding – with AI a well-educated person needs to be able to pose problems computationally and use good abstractions



Or be at the mercy of systems they can't evaluate

- Verification skills are important for AI today

Including using AI with verifiability in mind

MIT Schwarzman College of Computing