Intro Computing in 2035: *We've leveled up!*[*]

Aditi Gargeshwari[1], Bela Reis[1], Claire Wang[1], Heidi Repp[1], Karis Peebles[1], Kevin Wang[2], Vu Trinh[3], Zach Wood[1], and Zachary Dodds[1]
[1]Harvey Mudd College; Claremont, CA 91711
[2]Stanford University; Stanford, CA 94305
[3]Claremont McKenna College; Claremont, CA 91711
agargeshwari,breis,clawang,hrepp,kpeebles,vtrinh,zwood
dodds@g.hmc.edu,kevjwang@stanford.edu
[2]Computer Science Department

Abstract

Having returned from a decade hence, we share some heartening – and worrying – changes that Introductory Computing will be accumulating over the next ten years. On the plus side, both student and instructor agency – and resources – have increased dramatically. Standardized and solely-software CS curricula, on the other hand, play a much reduced role. The most valued skills were (1) *self-directed* new-system learning and (2) novel problem solving *across varied contexts*, with both of these (3) comfortably communicated, in person.

Based on what we've seen, in this work we share our current progress toward implementing 2035's Intro Computing mix. That mix includes creative, open-ended system-building, individual system-vouching and software-auditing, self-monitored skill-building, and face-to-face assessment. Our initial attempts, we realize, do not capture all the opportunities available, and this incautious approach is designed to maximize community feedback. This feedback, we believe, is the most important ingredient in aligning our curriculum-to-be with the class of '35's future needs. Our goal is that, together, all of us in higher ed will make the most and the best of what the next decade has in store!

[*]Copyright ©2025 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

1 Background: Our Future Computational Selves

Culture and computing have been co-evolving for as long as they have both been around [2, 3]. The decade ahead is poised to continue this tradition, perhaps even more precipitously [1].

For its part, higher ed seeks to increase knowledge, savvy, and adaptability by leveraging computing in service to humanity's shared cultures.[1] This looks quite different for introductory computing in 2035 than in decades past:

- In 2035, specifying-and-assessing-processes is no longer IntroCS's core. Core instead is *growing new cognitive models* for expressing and assessing processes.[2] Which is to say, **We've leveled up!**
- IntroCS mixes hands-on, how-it-works insight across far more computational models, e.g., string-passing via the web, voltage-passing via wires, meaning-passing via embeddings, value-passing via `return`, among others. We're seeing and celebrating **the water in which we swim**.
- Professionally speaking, the center has not held. Demand is outside-in: computing has seeped even deeper into cultural and corporate practices. These **capillary-level contexts** are where value is sought and communicated. *Computing-sans-context* has gone the way of *thinking-sans-context*, in terms of both personal and professional value.

2035 *does* share one feature with previous eras: everyone is born to communicate, but no one is born to compute.[3] Where computing is of value, interventions – such as IntroCS – will be needed to tap it. Next, we share the paths we are currently designing to tap that value:

2 Visualizations: Always Valuable, Newly Accessible

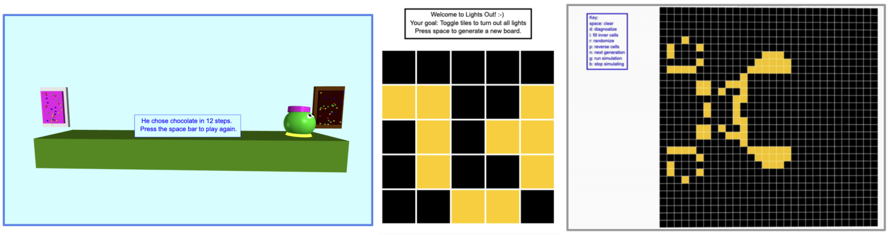
CS, and especially IntroCS, has always valued visualizations. They can, so often, effectively bridge human thought and computational workings. Given the syntax-supporting and summary-suggesting capabilities of our toolsets, student-authored visualizations have never been more accessible than in 2035. To realize these opportunities, we are now expanding our use of the `vpython` 3d library, of datavis via `matplotlib` and `seaborn`, and of purpose-built webapps, expanding the computational models IntroCS explores.

[1] We acknowledge that higher ed may have other purposes, as well.
[2] Perhaps this isn't a change at all, but it certainly **feels** like one.
[3] *Homo sapiens'* neurons may disagree, but they're not doing so consciously... .

VPython is an object-oriented Python graphics library which used to be a final-lab/final-project option for us. Now VPython appears in week 2, accompanying an assignment that models random walks. The "walker" is first visualized in the terminal. From there, students copy their code into a prepared VPython file, and see their implementation played out in 3d.

This assignment strongly emphasizes the 2025 experience of programming, formal and informal. That is, receiving and adapting near-complete code from resources like LLMs, often with little background knowledge. DIY-integration like this builds confidence and independence – and sophistication with LLMs as a resource.

VPython is leveraged in two more assignments: a game of LightsOut! implemented at first in the terminal and a simulation for Conway's Game of Life. These experiences smooth the final-prject option of creating a VPython user-interaction or game. This allows students interested in graphics to explore the library based on their own interests and solidifies their ownership and agency in the code.


He chose chocolate in 12 steps.
Press the space bar to play again.

Welcome to Lights Out! :-)
Your goal: Toggle tiles to turn out all lights
Press space to generate a new board.

Key:
space: clear
d: diagnoalize
i: fill inner cells
r: randomize
p: reverse cells
n: next generation
g: run simulation
b: stop simulating

Figure 1: Our additions of VPython include (**left**) a 1d walk animated with 3d scene objects, (**middle**) a 2d interface to *Lights Out!*, and (**right**) user-controlled simulation of Conway's *Game of Life*. All such examples expand students' *executable experience*, regardless of LLM use.

Seaborn offers an onramp to the data-science subset of intro computing. As computing edges toward becoming undergraduate-universal, so too will data science skills. (Data scientists, naturally, trace this causality in the other direction.[4])

Either way, the intuition developed by distilling processes into datasets, models, and graphs is central to many academic majors and many professional paths. We introduce Seaborn, like VPython, by integrating it into preexisting homeworks, each highlighting a different kind of graph: π-estimation by "throwing darts" at a unit circle inscribed in a square, aggregating and present-

[4]We CS'ers note that execution *is* causation, even if correlation isn't.

ing results within the random-walk and game-of-life assignments (rather than animating their evolution, for which VPython is better), and plotting signals in a lab that explores audio transformations.

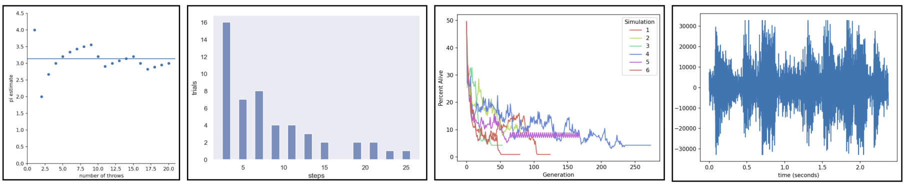


Figure 2: Seaborn makes data visualization smooth, enabling students to create their own summaries of (**left**) converging to π by dart-throwing, (**middle left**) the lengths of many fixed-radius random walks, (**middle right**) live-cell population sizes in Conway's *Game of Life*, and (**right**) transforming audio signals as lists of raw pressure samples. All students, *especially* those in data-dependent disciplines outside of CS, benefit from this practice.

After completing these assignments, students have seen their code come to light through provided graphics skeletons and then pick through the provided code to personalize their graphics interfaces. This inside-out style of learning helps students (1) realize they have full control of their software, (2) create a visual portfolio to showcase for future audiences, and (3) stay motivated to understand their code while still taking full advantage of AI tools.

3 Files+Folders: *Unhiding* our computing systems

Our sibling departments – especially in the natural, engineering, and axiomatic sciences – have long asked IntroCS to introduce the command line and comfort with the containment-hierarchy of files and folders. This is even more true when leveraging AI, since LLMs speak the language of the file system.

We thus believe additional practice is valuable, and this new unit reinforces the shell commands `pwd`, `ls`, and `cd`, and introduces a number of additional commands and concepts alongside that foundation. To make what can seem an esoteric experience both hands-on and memorable, we *customized* a fun, but generic, command-line adventure so that it was set on our campus. Folders represent places, e.g., nesting buildings and rooms, and their text files contain descriptions providing context and clues. Following feedback from other departments and majors, we embedded several novel skills and experiences:

- We included `cat` and common command-arguments, e.g., `ls -la`
- With `ls -la`, we then added "hidden portals," i.e., dot files/folders.

- Because the campus is our setting, every student is invited to *add their own room* to the adventure.
- Students can then contribute their locations on campus through the public repository on Github. Eventually we hope students will expand the adventure to cover all of campus and add their own secret folders and further terminal commands. Indeed, exercising this level of *individual agency* is a central goal of the experience.
- We integrated `ssh` into this adventure, with some of the clues hosted elsewhere, Remote login via command-line is another skill often requested by our fellow lab sciences (among others).
- In order to find the dorm and room with the treasure, student also practice executing command-line scripts, also widely requested. Because there are hundreds of distractor files and rooms, it would be nearly impossible to try opening every room to find the treasure. (We don't mention `cat */*/*.txt`, but we are *thrilled* if such "shortcuts" make the rounds!)

This scavenger hunt-style activity not only reinforces command-line proficiency, but it also serves as an informal introduction to the campus – an added benefit given that most students in the course are first-year students!

4 Hardware: *Unboxing CS*

Comfort with the command-line and filesystem does more than offer a shared language. It also establishes an agency over and understanding of our own computational processes. With details so readily available, it is this "big-picture perspective" that becomes the central value of IntroCS. For example, our IntroCS has traditionally introduced *simulated* circuits in order to reinforce that there is "no magic": computational interactions are explainable and understandable "all the way down."

To ground this understanding, we have expanded the course's opportunities from simulated hardware to physical hardware components such as transistors, resistors, and integrated-circuit gates. The majority of IntroCS students choose to major in engineering, and these experiences directly speak to their priorities. (In addition, these challenges tend to pique the curiosity of *all* students, because they are both so fundamental – and so rare!) At the beginning of the week, students pick up a kit with adequate materials, and, throughout the week, construct these circuits with the help of the lab guide:

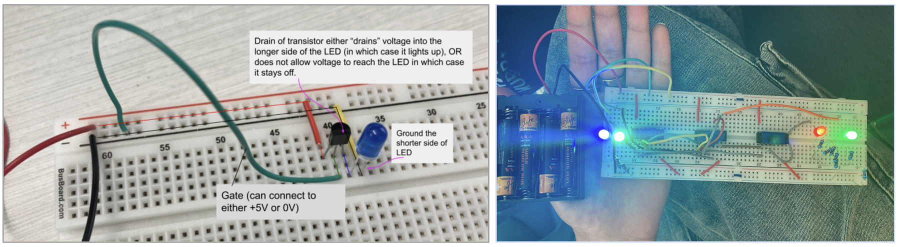

Drain of transistor either "drains" voltage into the longer side of the LED (in which case it lights up), OR does not allow voltage to reach the LED in which case it stays off.
Ground the shorter side of LED
Gate (can connect to either +5V or 0V)

Figure 3: By checking out low-cost components for a week, students (**left**) establish transistor behavior (and build a NOT-gate from two transistors) and (**right**) create a half-adder from standard AND, OR, and NOT gates. With an Arduino this single half-adder computes *arbitrary* additions, illuminating each bit, as it's added, via LED! (Python handles only the *storage* of the bits.)

4.1 Microcontroller hardware

With Figure 3's circuits, students use hardware to *implement* computation. We believe an equally important experience – and widely applicable professional skill – lies in using special-purpose hardware to *augment* computing's reach. From Professor Adam Blank's IntroCS course at Caltech, we have adapted a hardware unit built around the Adafruit *Proximity Trinkey* (Figure 4). The Trinkey is a USB-A programmable microcontroller equipped with an Adafruit Trinket M0 and an APDS9960 sensor. Since the Proximity Trinkey acts as a USB key, no cable is needed; it plugs directly into a USB-A port.

Our adapted assignment sequences microcontroller-interaction into three component assignments:

- NeoPixel LED Lab: Students program an LED strip using the QT2040 and a NeoPixel breakout board, introducing students to serial communication and driving LEDs.
- Adapted Morse Code Assignment: Students program the Proximity Trinkey to translate Morse Code into English in the serial port. This assignment also introduces students to interfacing with sensors: capacitive touch pads and proximity sensors.
- An HID Interaction Assignment: Optionally employing the Proximity Trinkey, Wii Nunchuk, or Mini I2C Gamepad, students use a Human Interface Device library to interact with their computer, e.g., by triggering keyboard commands to control gameplay in an existing app.

All instruction materials and starter code are provided in both CircuitPython and Arduino to accommodate varied student backgrounds and objec-

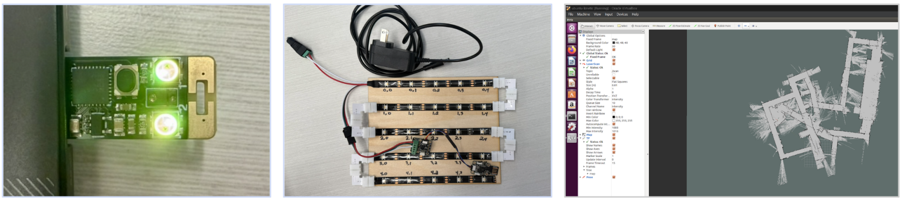


Figure 4: About the size of a quarter, the Adafruit Trinkey (**left**) is a USB-A device with available buttons, proximity sensor, and LEDs: we borrowed Prof. Adam Blank's elegant *Morse Code* assignment. As a lab, students can connect to pre-wired light strips (**middle**) to more dramatically illustrate their *Lights Out!* programs. Hardware interaction is empowering and, sometimes addictive. A discarded lidar, found and researched, yields maps of the engineering-shop hallways (**right**). This independent, open-ended research exemplifies the value of building a portfolio of AI-enabled interactions, proactively pursued and communicated.

tives. These microcontroller-hardware experiences are intentionally scaffolded with LLM support: the assignment invites students to proactively teach themselves, via strategic AI interactions, the skills needed to create and debug an exemplar system. A concluding deliverable reflects on this proactive process. It is this independent – and independently-directed – skill-building (as well practicing the confidence to try it out) that are the most important learning objectives of this hardware-themed unit.

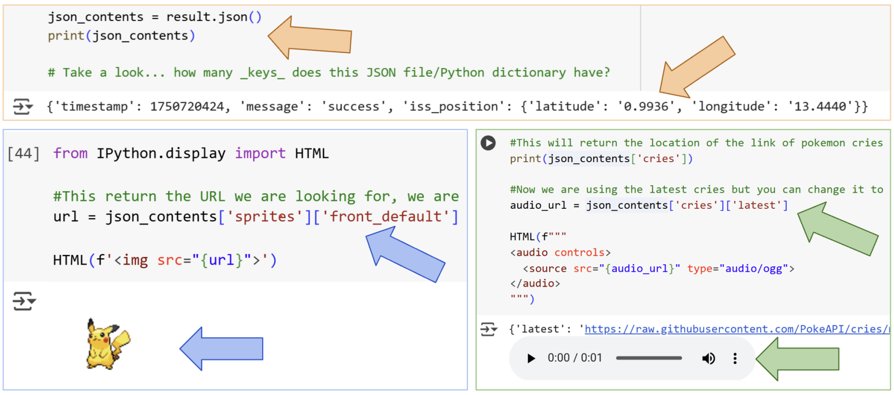
5 Program-defined message-passing: APIs

In 2035, APIs have become a foundational component of IntroCS.[5] Whether measured economically or culturally, API calls are *the* most important software interaction. The coming decade's AI-enabled programming-support both epitomizes the centrality of API use – and offers the support needed to support their exploration and use from computing's get-go. Happily, there is a natural evolutionary path for API calls: the data they return is a wonderful and practical application of Python dictionaries. This section will outline how APIs can be integrated into a CS1 curriculum.

In our introductory computing course, students first use the GET method to collect and interact with data from the International Space Station (ISS) API, Pokemon API, and Wikipedia API. The ISS API is the starting point because of its relatively readable response, which helps students feel more comfortable

[5]One could even argue this is a decade or two late!

working with APIs. The students will then practice retrieving specific information within the response. This exercise will help students practice dictionary look up, an essential skill in working with API. Their next task will be to work with the longitude and latitude returned by the API and to use a function provided to calculate the distance from the ISS to a city of their choice.



```
json_contents = result.json()
print(json_contents)

# Take a look... how many _keys_ does this JSON file/Python dictionary have?
```

```
{'timestamp': 1750720424, 'message': 'success', 'iss_position': {'latitude': '0.9936', 'longitude': '13.4440'}}
```

```
[44] from IPython.display import HTML

     #This return the URL we are looking for, we are
     url = json_contents['sprites']['front_default']

     HTML(f'<img src="{url}">')
```

```
#This will return the location of the link of pokemon cries
print(json_contents['cries'])

#Now we are using the latest cries but you can change it to
audio_url = json_contents['cries']['latest']

HTML(f"""
<audio controls>
  <source src="{audio_url}" type="audio/ogg">
</audio>
""")
```

```
{'latest': 'https://raw.githubusercontent.com/PokeAPI/cries/
```

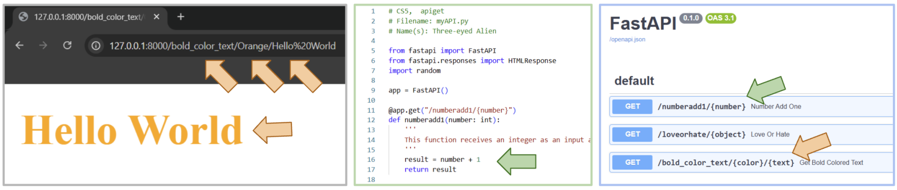
Figure 5: API calls naturally reinforce foundational programming concepts, e.g., dictionaries' key-value pair organization, while offering authentic opportunities to individualize early-software experiences, e.g., by determining the current location (and, from there, distance) to the ISS (**top**), by programatically rendering a particular Pokemon (**bottom left**), even obtaining and playing its unique cry! (**bottom right**)

For many, the Pokemon API offers an even more customizable and relatable challenge. Unlike the ISS API, the Pokemon API returns a much more *authentically complex* dictionary (via JSON) with dozens of keys and subkeys, making the data-selection process more demanding. This is a crucial challenge, and one that AI tools are well-poised to assist – **if** a user knows what they are looking for and what to expect in return! Students also experience – and control – how APIs can return a variety of data formats, like images and audio files – of Pokemon, even!

5.1 APIs as "real-world" software

Students follow these experiments with the Wikipedia API, which cleanly illustrates how every webservice and website are simply renderings of API calls – and, what's more, that they are able to *programmatically* access those calls and results! The Wikipedia API also offers direct Python support, unlike the ISS and the Pokemon APIs, so that students experience some of the wide variety across API use.

An exciting - and professionally relevant - facet of API use is designing, deploying, and testing one's own API. Using FastAPI, students first develop and run small-scale APIs locally on their own machines. This is a valuable experience that helps students deepen their understanding that "the cloud" is just other people's computers, and how its data is accessed using the GET method. As students build their APIs, they define multiple endpoints and use query parameters to filter or customize the data returned by the API. Recognizing – and controlling – the components of a URL underscores IntroCS's primary goal: students' hands-on experiences of agency within the ambient oceans of computational processes.


127.0.0.1:8000/bold_color_text/
127.0.0.1:8000/bold_color_text/Orange/Hello%20World
Hello World

```python
# CS5,  apiget
# Filename: myAPI.py
# Name(s): Three-eyed Alien

from fastapi import FastAPI
from fastapi.responses import HTMLResponse
import random

app = FastAPI()

@app.get("/numberadd1/{number}")
def numberadd1(number: int):
    '''
    This function receives an integer as an input a
    '''
    result = number + 1
    return result

```

FastAPI 0.1.0 OAS 3.1
/openapi.json
default
GET /numberadd1/{number} Number Add One
GET /loveorhate/{object} Love Or Hate
GET /bold_color_text/{color}/{text} Get Bold Colored Text

Figure 6: Building one's own API conveys the agency we have over the computational interactions of contemporary experience, e.g., HTML pages as human-readable results of API calls (**left**), the ease and power of making small, self-authored functions API-accessible (**middle**), for which our approach is the well-documented, widely-used FastAPI library.

The API-creating exercises are scaffolded to reveal these connections with familiar artifacts (HTML pages). For example, the provided starting point is a function that receives a color and a string (as URL-path components), and their API returns a small HTML page, in bold and in that color. From there, students create a new, more complex API where they will create a dictionary, ultimately achieving behavior similar to the ISS, Pokemon, and Wikipedia APIs used as initial examples.

FastAPI offers benefits beyond insight. As an industry standard, it automatically generates documentation for the APIs students create, inviting (course-required) descriptions of what each endpoint does. This reinforces the importance of API documentation, for both author and caller. The ability to read and interpret documentation is a crucial skill, as it enables students to learn new libraries and APIs independently. In this way, creating an API with FastAPI equips students with a practical tool for self-directed learning.

6 A Preface, not a Conclusion

Contemporary forces have transformed computing from a “valuable specialty” toward a “universal literacy.” This is both a challenge and an opportunity for us as computing educators. On one hand, our audience is growing rapidly – very few institutions and disciplines consider computing an *optional* experience in 2025. That number will likely be fewer in 2035.

Yet the *content* of undergraduate-universal computing is changing at least as rapidly. LLMs enable our students to be dramatically more efficient and effective in exploring and authoring computational processes. Given computing’s universality, computing education’s responsibility is no longer tied to supporting specific professional paths (if it ever was). Instead, the IntroCS experiences of the decade ahead will offer a hands-on mix of self-directed project-building and -presentation.

To be sure, there will still be a need for syntactic sophistication. There will, we hope, still be CS departments enthusiastically expanding the “magic” that converts intent to source, source to semantics, and semantics to reliable value.

But in 2035 AI will *precede* CS, not follow it.[6] In disciplines or institutions where *every* student engages with computing, it is crucial that the content of Computing1 be structured in a way that serves students *independent* of the specifics of their future professional paths.

This work highlights our initial, awkward efforts toward a universal experience of computational ownership and agency. To this end, we look forward to both feedback and pushback as part of linking efforts across higher ed. The coming decade offers us all an “unavoidable opportunity.” We look forward to tackling it – together.

Acknowledgement

The authors gratefully acknowledge resources from NSF DUE # and from Google, as well as support from our respective institutions.

References

[1] Brianna Blaser. “RESPECT 2025: Designing an Accessible Future for Equitable Computer Science”. In: *SIGCSE Bull.* 57.2 (Apr. 2025), p. 4.

[2] Mark Guzdial and Daniel Reed. “Securing the future of computer science; reconsidering analog computing”. In: *Commun. ACM* 56.4 (Apr. 2013), pp. 12–13.

[6] We note this is already true, lexicographically...

[3] Markeya S. Peteranetz et al. “Future-Oriented Motivation and Retention in Computer Science”. In: *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. SIGCSE ’18. Baltimore, Maryland, USA: Association for Computing Machinery, 2018, pp. 350–355.