Engaging with the Wiphala: A Code-Generation Hardened CS1 Final*

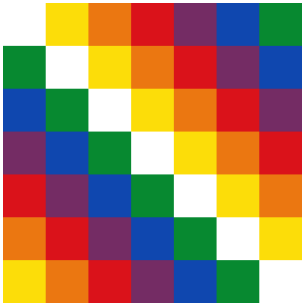
Calvin Deutschbein and Shouvik Ahmed Antu
Willamette University
Salem, OR 97301
{cdeutschbein, sahmed}@willamette.edu

Abstract

As co-instructors in introductory computing classes at a small college, in Fall 2024 we needed an approach to code generation tools that at once equipped new students to be effective in a world in which those tools existed, but also still benefited from the computational thinking and literacy skills often covered in the first semester of a computing course. Many introductory computing assignments, including some subset of SIGCSE Nifty assignments, remain perhaps too challenging for many introductory students even when specified to such a level that freely available code generation tools, such as ChatGPT or Gemini, are able to produce perfect solutions. However, there remain many tasks that are quite difficult for code generation tools, which by construction are specialized to text analysis and text generation. In our Fall 2024 introductory courses, we deployed an experimental final for which the prompt was to generate a certain form of image informed by cultural context to communities adversely impacted by the surge in data center demand in the U.S.. Students would necessarily analyze and describe an image, and critically engage with the consequences of over-reliance on code generation, in order to complete the assignment. We achieved our desired grade distributions and believe we differentiated students who were learning material from students who were outsourcing coursework to automated tools.

1 Introduction

Figure 1: The Wiphala is a square emblem commonly used as a flag to represent some native peoples of the Andes.



At our university, the entry point for computer science education is "CS 151: Introduction to Programming in Python", which is a Python translation of an older class in Java based on a Stanford curriculum leveraging SIGCSE Nifty assignments such as Adventure Game [5], Breakout [6], Enigma Machine [7], and Wordle [4]. These are established assignments, with years of success training new computer scientists, and extremely in-depth instructions provided directly to students and available widely online. Coincidentally, making these assignments more accessible for students has also made it easier for students to not engage with these assignments at all due to the availability of ChatGPT, Gemini, and similar frameworks capable of code generation. We perceive a growing consensus that reliance on such tools hampers cognitive development [3] and therefore should be addressed in course design.

In this paper, we will present our novel final on rendering the Wiphala (Fig 1). We believe this final supports student learning and allows students to engage with code generation and its limitations.

We are motivated by a particularly troubling phenomenon. The CS1 curriculum leverages a Python port of the Portable Graphics Library "PGL" [8] which often confounded earlier code generators expecting PyGame or similar libraries. However, specifically in the case of the Breakout assignment, both ChatGPT and Gemini generated correct solutions without being provided a description of the Python PGL library while struggling on more general tasks. We suspect exact solutions to Nifty assignments are present within training data in some capacity.

Nifty assignments are unified in having clear (1) learning objectives and (2) assignment formats, which we can mimic for novel assignments intended to defeat code generation. Our final has the following learning objectives:

1. Fundamental Python
2. Reading Python
3. Strings and Files
4. Interactive Graphics
5. Defining Classes

As a rule, "Strings and Files" maps to the text-based Adventure Game, "Interactive Graphics" maps to graphical Breakout game, and "Defining Classes" maps to the "Enigma Machine" ciphers. "Fundamental Python" and "Reading Python" are expected to be manageable by students with debugging experience. We mapped these objectives to what we expect to be AI-hard subproblems, synthesized subproblems into a coherent assignment, and applied the assignment to a culturally relevant application domain.

We found:

1. Our approach achieved a grade distribution consistent with pre-code generation classes and our personal assessment of student engagement, while allowing use of code generation.
2. Our approach maintained the core learning objectives of an ACM 2013 [2] CS1 course.
3. Our approach equipped students to understand the benefits and limitations of code generation.
4. Our approach raised critical questions about the broader social impacts.

We have made a generalized form of this assignment (replacing all dependencies with PyPI alternatives) available publicly on GitHub [1], with web-hosted instructions[1], available under MIT License for all code artifacts and with fair use images documented with source attribution. We will provide solutions by request to verified instructors.

2 Format

Our introductory course was structured across lectures, problem sets, and projects but the primary learning content is delivered via the projects - the four (4) Nifty assignments and a fifth pseudo-Nifty assignment "ImageShop", a rudimentary image editing program.

[1] `https://cd-public.github.io/ccscrm/`

We regard the first three projects as *scripting* assignments. They are assigned over the course of one week and completed in one file (or module). For these assignments. Students write functions which interact with a provided template. The final two projects - "Enigma Machine" and "Adventure Game" were *programming* assignments spanning multiple weeks, multiple files, and potential multiple team members. In Fall 2024, average dropped sharply from the scripting assignments (averages of 84, 87, 87 percent) to programming (averages of 79, 69 percent). We believe this drop may be associated with lacking the technical skill to prompt a code generator for more than isolated functions. Therefore, we administered a highly interdependent final exam.

Each assignment was split into "milestones" well suited to daily work. For example, the "Wordle" assignment was released Monday, had six zero-indexed milestones and was due the next Monday, giving students a final day after expected completion to debug any last-minute errors. This milestone structure and its familiarity to students supported a similarly structured final with multiple learning objectives as objectives could be mapped to milestones.

We assigned problem sets and projects via GitHub classroom as a code repository and separately provided instructions in HTML. In response to student feedback, we organized the assignment descriptions into one HTML `<details>` block per milestone.

3 Learning Objectives

We will describe a learning objective, how widespread availability of code generators has impacted assessment, and our solution.

3.1 Strings and Files

Existing finals prompted students to, for example, find the first numeric digit in every line of a file containing lines of ASCII characters. This is trivial for code generation or programmers familiar with library functions.

As an alternative, we ask students to read the names of colors and their hex values from a file. Critically, we introduce what we term *intentional ambiguity*, that is, we do not specify the format in the file or the format which will be most useful for latter milestones. Rather, students must design and then implement a solution. While we provide a CSV file, and reading CSV files into Python is trivial, we pose a number of complexities:

```
rojo 219 10 19
naranja 236 120 8
amarillo 252 222 2
blanco 255 255 255
```

```
verde 1 138 44
azul 6 69 177
violeta 117 40 100
```

We mix alphabetical and numerical values, provide no column labels, provide non-English color names, and space-separate rather than comma-separate. We refer to these as "red-green-blue" color descriptions in the text but otherwise expect students to either know or be able to determine how to work with color data in this format. Students also must independently determine whether, for example, to return multiple lists of strings and integers, or a dictionary of lists, or any other solution. We saw numerous possible solutions within high-scoring final submissions, and are pleased to have perceived space for students to make design decisions.

3.2 Interactive Graphics

Existing final examinations prompted students to, for example, render a pie chart from a user-provided list of percentages. This is obviously trivial to code generators via library functions or rudimentary geometry.

We ask students to generate a Wiphala (Fig. 1). It requires listing colors in the same cycle, but with a different starting point, across rows (or columns). We believe that it is of comparable difficulty to write a loop- or slice-based solution as to instruct a code generator with sufficient precision. Therefore, we feel students necessarily approached the computational thinking problem of multidimensional indexing. Naive generated code often produced a bottom-left-to-top-right diagonal, which we found among low-scoring submissions.

Separately, we take great care to avoid referring to the Wiphala by name in the instructions, as this is often sufficient prompting for a code generator to produce competitive solutions. We aim to resolve this tension in future work, perhaps with a novel application domain.

Lastly, we take care to inform students what the flag represents - the self-determination of the Andean people - with some allusions to the political implications of U.S. tech industry expansions on mineral and labor rights in the Andes. We hope that students carry with them a sense of the true cost of relying on exascale compute clusters for trivial tasks like list/array slicing as they continue their careers.

3.3 Reading Python

Traditionally the final exam began with a code tracing exercise. For example, we may ask a student what the following returns:

```
def conundrum():
```

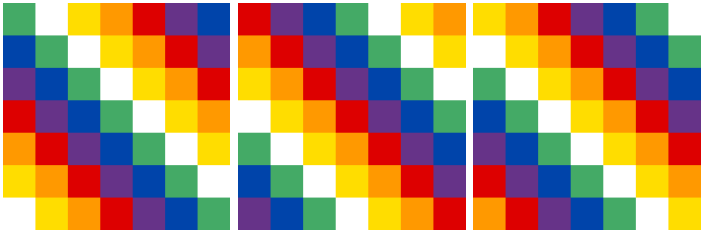


Figure 2: The Wiphala of other Suyu (Anti, Chinchay, and Kunti).

```
array = []
for i in range(6):
    array.append(0)
    for j in range(i, 0, -1):
        array[j] += j
return array
```

We believe we have leveraged the Wiphala to achieve a similar goal. In the prior section we show a specific, regional Wiphala, for the Qulla Suyu (Qulla region). Other recognized regional Wiphalas are differentiated by "long horizontal" color as shown in Fig. 2. Rather than prompt students as to the outcome of code, we prompt students to write code that generates some outcome specified with imagery, rather than text, which may pose a barrier to code generation. Additionally, we manually render the flags as fixed-pixel HTML elements rather than providing an image that can be pasted into a code prompt.

A small number of students entertained manually specifying all 49 tiles in the earlier Wiphala, and this subproblem encouraged use of programming logic.

3.4 Defining Classes

Existing final exams often ended with a larger task involving implementing a `class` that, for example, created numbered labels or recorded the dot distribution of a domino. Given Python iterators and collections, respectively, and expect students will often instead interact with classes implemented within libraries, such as PyTorch tensors, or Flask APIs. While we have a University-specific implementation utilizing classes, we generalize to a NumPy `ndarray`.

We ask students to (1) read file data based on file names provided in the CSV file from an earlier problem, (2) interpret these files as images, (3) crop images to squares, and (4) scale images to be the same size. Finally, rather than creating a Wiphala of colors, they create a Wiphala as a collage over these images as shown in first image of Fig. 3. These are fair use images

inspired by each individual color. For example, we selected an image of a Bolivia lithium mine as "el símbolo de las riquezas naturales" (the symbol of the natural resources), represented by the verde color in the Wiphala, and provide a README with this note and source attribution.

We retain a small amount of *intentional ambiguity*. We could define this operation in pseudo-code: if the pixel at location `(x, y)` would be of color "amarillo", write the "amarillo.png" value at this index to the output pixel array. Rather, we provide example output and a brief recommendation to visually inspect the long diagonal and the constituent files. The distribution of final scores suggests few-if-any students reaching this problem were confused by these instructions.

Figure 3: True-color and color-scaled collages, both provided to students.

The collaging effect appears to confound the image processing capabilities of most code generation tools. Gemini inferred it was observing a 5-by-10 grid of 50 individual images with no pattern, even when prompted to use 7 provided images. ChatGPT inferred it was provided with single image file of 7 concatenated tiles and was trying to create a 10-by-10 grid.

3.5 Fundamental Python

In the context of the final exam, the term "Fundamental Python" is used to refer to what is often termed numerical computing, like computing the Racamán sequence. We used NumPy vector operations, the foundation of early numerical computing efforts in Python. For each image, after cropping and scaling, we ask students to convert the true-color images to monochrome. We term this "colorscale" to remind students of grayscale from an earlier assignment ("ImageShop"). To support them, we specify:

1. Example code to compute luminance:

```
r, g, b = color; (4*g + 3*r+b)//8
```

2. That high luminance should become “rojo” monochrome value.

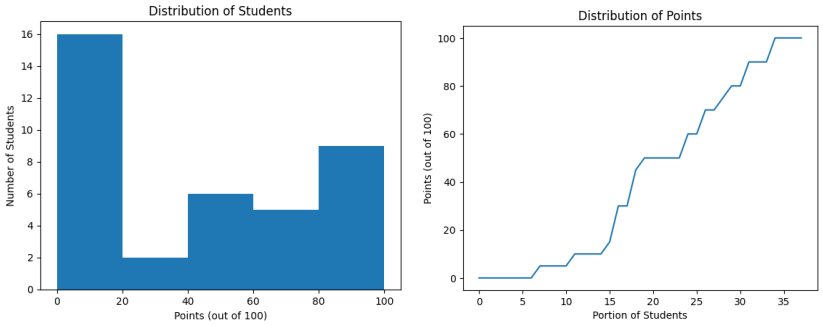
3. That low luminance should become all zeros (black).

Of note, we are not aware of particularly succinct ways to describe this process absent color science terminology outside an introductory computing scope. Separately, we provide a vector-optimized integer luminance calculation to partially confound code generation tools, which seemingly expect to perform these calculations over floating point values. We provide the second image in Fig. 3 as an example to our students with the caveat that their code must additionally be able to produce similar images for the other regions.

4 Results

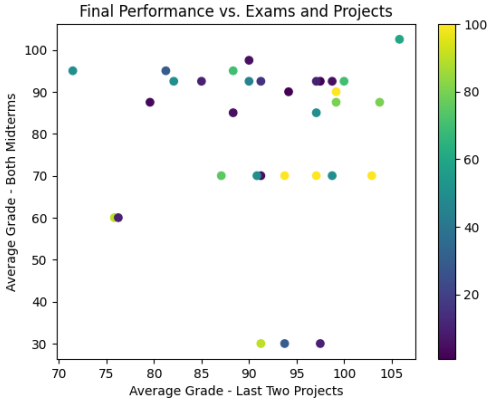
We achieved our preferred point distribution of approximate uniformity, allowing us to differentiate student achievement. We note we applied a generous curve (not shown). An uncomfortably large number of students earning zero or nearly zero points. Anecdotally, from in-class exercises and office hours, we suspected a population of students earned high marks on assignments but could not write a single line of code, even given hours to do so. We suspect this population gave rise to the large bin of low scores shown in Fig. 4.

Figure 4: We achieved a quasi-uniform distribution.



We found the final score uncorrelated with project or midterm scores, as shown in Fig. 5. We plot horizontal homework (code generation allowed), versus vertical midterm (code generation disallowed but suspected), versus colorized final score (code generation unhelpful).

Figure 5: Project (horizontal) and midterm (vertical) performance did not predict final (colorized) performance.



5 Summary

Our impression was that highly motivated students were completing exams and projects manually with minimally but non-zero errors and being scored lower than students who solely used code generation, including in violation of midterm honor policy. Many high scores on the final were students we identified as strong scientists, but lacked interest to complete portions of assignments they did not find algorithimcally interesting. Conversely, the final seemed to firmly differentiate an otherwise co-mingled population of relatively high-scoring students between independent and assisted coders. With some relief, we do not seem to be measuring testing anxiety given the low correlation ($r^2 = .0057$) with midterm scores. That said, we want to be more confident we are measuring student learning and not some other factor, and hope to deploy similar midterms and in-class exercises in Fall 2025 for students to be better prepared for the eventual final.

Ultimately, we believe this final rewarded hard-working students. We also hope to have persuaded our class of the usefulness of course content even in an environment with widespread access to industry-grade code generators.

6 Special Thanks

We profusely thank Professor Jed Rembold, our CS 151 co-instructor, for providing thoughtful feedback, showing patience during development, and maintaining all five existing projects, among many other contributions..

References

[1] Calvin Deutschbein. *Final*. https://github.com/cd-public/ccscrm. 2025.

[2] Association for Computing Machinery (ACM) Joint Task Force on Computing Curricula and IEEE Computer Society. *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. New York, NY, USA: Association for Computing Machinery, 2013. ISBN: 9781450323093.

[3] Nataliya Kosmyna et al. *Your Brain on ChatGPT: Accumulation of Cognitive Debt when Using an AI Assistant for Essay Writing Task*. 2025. arXiv: 2506.08872 [cs.AI]. URL: https://arxiv.org/abs/2506.08872.

[4] Nick Parlante et al. "Nifty Assignments". In: *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 2*. SIGCSE 2022. Providence, RI, USA: Association for Computing Machinery, 2022, pp. 1067–1068. ISBN: 9781450390712. DOI: 10.1145/3478432.3499268. URL: https://doi.org/10.1145/3478432.3499268.

[5] Nick Parlante et al. "Nifty assignments". In: *Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education*. SIGCSE '02. Cincinnati, Kentucky: Association for Computing Machinery, 2002, pp. 319–320. ISBN: 1581134738. DOI: 10.1145/563340.563466. URL: https://doi.org/10.1145/563340.563466.

[6] Nick Parlante et al. "Nifty assignments". In: *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education*. SIGCSE '06. Houston, Texas, USA: Association for Computing Machinery, 2006, pp. 562–563. ISBN: 1595932593. DOI: 10.1145/1121341.1121516. URL: https://doi.org/10.1145/1121341.1121516.

[7] Eric Roberts and J. Jedediah Rembold. "Nifty Assignments: Enigma Machine Simulator". In: *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 2*. SIGCSE 2023. Toronto ON, Canada: Association for Computing Machinery, 2023, p. 1276. ISBN: 9781450394338. DOI: 10.1145/3545947.3573220. URL: https://doi.org/10.1145/3545947.3573220.

[8] Eric Roberts and Keith Schwarz. "A portable graphics library for introductory CS". In: *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education*. ITiCSE '13. Canterbury, England, UK: Association for Computing Machinery, 2013, pp. 153–158. ISBN: 9781450320788. DOI: 10.1145/2462476.2465590. URL: https://doi.org/10.1145/2462476.2465590.