Pre-Introductory Programming for College Students: Driving Engagement, Motivation, and Creativity to Drive Interest in Computing Studies

Andras Magitay-Becht
Economics
Saint Mary's College of California
Moraga, CA
am17@stmarys-ca.edu

Udayan Das
Computer Science
Saint Mary's College of California
Moraga, CA
udd1@stmarys-ca.edu

***Abstract*— This innovative practice paper presents an approach to make computer science (CS) in general and programming in particular more approachable for college students. Introductory programming classes can be difficult for learners; especially those without any programming background. Beginners must learn a new way of thinking, made more difficult by issues learning syntax. It's especially hard for students who lack a strong mathematics background. Observations across 3 editions of a CS0 pre-intro-programming course show that students consistently overestimate difficulties, and, come away with a stronger motivation for studying programming in the future as well as considering computer science (CS) and data science (DS) majors, minors, and certificates as options for future study after completion. A wider variety of students consider CS-adjacent college programs, and this can have an appreciable impact on the diversity of the student body entering CS/DS programs. We have found that long term success in CS/DS is predicted by strong motivational factors. While many high school students bring a strong motivation for pursuing CS/DS due to socioeconomic factors (exposure through people they know) and availability of coursework in high school; for other students, some may not have considered CS/DS as an option due to perceived difficulty as well as steep math requirements that function as barriers-to-entry. A CS0 pre-programming course like ours explicitly centers itself as "coding-is-fun" and encourages creativity. Students engage strongly with and are motivated to put in extra outside-of-class work required for mastery of material. We have limited evidence due to the fact that the data we have is from our small liberal arts college, however, 2 key factors make our observations and analysis valuable: 1) being a Hispanic-Serving-Institution (HSI) and an Asian-American and Native-American Pacific-Islander Serving-Institution (AANAPISI) we have a diverse student body, and 2) we are not a highly selective college which brings in a wide variety of students in terms of prior preparation. Having a CS0 pre-programming "coding-is-fun" allows this diverse student body to "test-the-waters" and, if interested, to pursue further CS/DS education by reducing the barrier-to-entry. We present data on: student perceptions before/after taking the course; the numbers of students considering further study including the breakdown of course-based vs self-study; and the tremendous numbers of students who find coding easier and therefore something they will use in their future life regardless of whether they formally plan to study CS/DS. The final point is critical in terms of adjusting to life in the 21st century as many fields/professions continue to increasingly be influenced by information and data technologies.**

Keywords— CS0, Intro Programming, CS4all, CS education

I. INTRODUCTION

Since January 2023 we have run three iterations of a pre-introductory programming class built around the same base concept. While the core and intention of these classes were the same, the actual execution, target audience and student reception differed.

A. Base concept

The key idea behind the classes is to provide an equitable background to programming-interested students coming from computationally deprived areas. We are re-creating a condensed form of the introduction to programming experiences well-funded middle and high schools can provide their students.

1) Key concept 1: focus on utility and fun. All classes were aiming to provide the students with instant gratification: immediate positive feedback between new material and enhanced creativity.

2) Key concept 2: just in time instruction. Instead of introducing new concepts directly and then providing the students with examples of how to use them, new concepts were introduced as an answer to a need from the students to be able to achieve a goal.

3) Key concept 3: start with block programming. Since the purpose of the classes is to focus on accessibility, utility and fun, and also to make up for the lack of middle- and high-school accessible programming experiences, all versions of the classes started with a significant Scratch section. This provided three benefits:

- Quick and easy results are achieved within minutes of starting the work. The students do not need to get excited about a "Hello World!", they can create their first animations in the first few minutes. This can easily be expanded to a full game already during the first class section, establishing a positive feedback loop between learning new material and creating new experiences faster than you could in a text-based programming language.
- No need to worry about syntax: the code blocks provide help for the students to see what information is required in order for the given block to operate. Want something to go on forever? Use a forever loop. It has no parameters, and it is impossible to add another code

block after it, reinforcing the otherwise intuitive idea that since a forever loop never quits, it would be pointless to add follow-up instructions after it. If, on the other hand, you only want to repeat something until a given condition is met, the repeat until loop clearly illustrates that it requires a boolean statement, since all boolean statements are represented by hexagons. Unlike the forever loop, the repeat until loop also contains a little tab at the end of it, to allow additional code blocks to be added to follow it.

- Encourages creativity. In the minds of some students and parents, and even in some academic communications, there exists a juxtaposition of creative jobs and careers, assumed to be tied to the arts like drawing, painting, digital media, creative writing, theater, dance, etc., versus non creative jobs This view of creativity is really narrow, and the use of Scratch or Scratch-like environments leads the students to the realization that coding is a creative endeavor as well.

B. Base course material and topics

Since we are defining our approach as a pre-introductory, CS0-like course, it is important to define what we mean exactly by that. We argue that a Scratch-based (or block-programming based) pre-introductory class doesn't just act as preparation for an Introduction to Programming course, but also provides the students with insights to more complex topics, preparing them for the entirety of their major. The easiest way to show this is to compare the material covered in a pre-introductory class with that of a CS 1 class. For this comparison we will use as baseline the CS 1 material described in [1] (see figure 1).

Material	CS 1	Pre-introductory programming
Variables	Yes	Yes
Data Types	Yes	No
Console I/O	Yes	*Limited*
Expressions	Yes	Yes
Branching / selection	Yes	Yes
Loops	Yes	Yes
Functions	Yes	*Limited*
File I/O	Yes	*Limited*
Modules and libraries	Yes	*Limited*
Objects	Yes	*Limited*
Recursion	Yes	No
Event-based control	No	Yes
Concurrency / Threads	No	Yes

Fig. 1. Concepts comparison between standard CS1 and out CS0 pre-introductory programming courses.

Variables, expressions, branching and loop are shared aspects of both introductory and our block-code based pre-introductory programming classes, while recursion and data types only appear in the introductory classes. Certain topics, however, appear in a limited fashion in block programming context that allows the pre-introductory experience to prepare the students for work in the intro to programming classes:

- Console I/O: While there is no console *per se*, various sprites can "say" things as an output, and the ask statement allows the user to input values to the program at runtime.
- Functions: Scratch has support for custom code blocks, but these are explicitly procedures and not functions, as they cannot return a value. Variables can be used to preserve the internal state of a custom code block after its execution has completed. Other block programming languages, like Snap!, however, allow proper functions to be created: the original name of Snap, Build Your Own Blocks, alludes to this fact [2]. We have deliberately chosen to use Scratch over Snap! to lower the barrier to entry for the students, but Snap! is a possible (and worthy) extension of the course (see below)
- File I/O: This is so limited in Scratch that it cannot truly be called as File I/O. But when one saves a Scratch program as a file, all of the current values of the variables get saved along with the code blocks themselves, so it is possible to save and restore the state of the given program. There are no explicit built-in ways of reading from or writing to a file, however.
- Modules and libraries: Scratch comes with a list of extensions that can be considered as using modules, though the user cannot create modules on their own. Some of these modules are introduced as integral parts of the base course, like the Pen module that allows the implementation of the old Logo Turtle drawing approaches, or the Video Sensing module that allows the students to control the program using a webcam through gestures. Other modules like the micro:bit and the various Lego modules allow for extensions of the course that connect the virtual world of programming with the physical world the students inhabit. This could be an appealing area of extension, but with a significant cost barrier. The existence of these modules allows the students to learn about the *concept* of extending the programming language itself, without having to worry about interacting with modules and libraries deeper.
- Objects (*lite*): in Scratch the students cannot really create new objects. However, Scratch is based on sprites and manipulating these sprites. Sprites

themselves are objects, with explicit attributes that can both be read from and written to. This prepares the students to understand and work with the abstract concept of objects in a very hands-on manner. At the same time, the users do not have the ability of adding new properties or methods to the sprites, and do not have to worry about visibility of properties, let alone *inheritance*.

In Scratch, execution usually begins with the *green flag* event. Students quickly learn that they can place multiple green flags to begin program sequences, and all begin executing at the same time. The introduction to this concept is integral to the material. If they have multiple sprites in their program, all of the sprites need to listen to the green flag event. Students realize soon that they can place multiple green flag starts even in the same sprite, so the concept of multiple threads arises naturally. From this, we can organically buildup to the realization that there might be a need to synchronize among these events, which leads to the introduction of the messaging code blocks. This early introduction to event-based control and concurrency/threads, is not usually seen in CS1 intro programming courses, typically being introduced later.

C. Scratch or Snap!?

We have investigated both programming languages as the baseline material for this class, and in the end decided to use Scratch. Snap! is an excellent extension of the base Scratch features, and is a great platform to build introductory programming experiences on. Indeed, the home school of Snap!, UC Berkeley, runs a course called "The Beauty and Joy of Computing" using the Snap! Platform [2].

We decided to start with Scratch because of its simplicity. The primary purpose of our classes is to be as gentle of an introduction as possible. We found that the various amazing features of Snap! like first class lists, first class procedures, ringed procedures etc. make the material harder to approach, so Scratch is a better fit for our student population.

D. Extensions

All three iterations of the course devoted a third- to a half of the class to Scratch-based basics. Since our target audience is university students, that much time is sufficient to master the basic concepts covered in Scratch, and the course can progress with extensions.

1) Computational thinking: while theoretically only an extension, all three iterations of the class contained significant class time building up these conceptual frameworks. We do consider them to be critical components highlighting for the students the universality of the problem solving approaches in computer science [3], [4]. The classes discuss how the steps in computational thinking and/or design thinking paradigms can help when preparing homework, or trying to write a book, compose a new piece of music, etc. Of particular interest was the tool of storyboarding, embraced by the students as a way of thinking through the project [5], [6]. These conceptual frameworks were also excellent for gap analysis. Students could identify skills they needed to work on to execute a particular project, which led to motivation for self-study and exploration.

2) Website coding and design: A highly popular extension among the students. They liked the quick positive feedback loop and near-instant gratification when working with WYSIWYG website design environments like Google sites or Github pages, but also enjoyed learning how HTML and CSS worked. This extension also enabled the students to create a portfolio of their own work, further enhancing their appreciation of the work they did in the class.

3) Spreadsheets and spreadsheet programming: since one of the authors comes from an Economics and Business education background, they are greatly familiar with the various spreadsheet environments. They believe that spreadsheets are an excellent way to learn some basic programming concepts. The built-in IF statements both in Excel and Google sheets allow the students to practice branching, and the different cells of a spreadsheet serve as nice analogs for variables. To integrate variables further and add loops to the material one can include VBA programming for Excel or JavaScript programming in Google Sheets. An advantage of this approach is that there is no need to worry about input/output processes since the code can write directly into the underlying spreadsheets, simplifying the material to cover. While the authors had some success with this approach in a different context, for a pre-introductory class attracting students from a large variety of majors this proved to be a highly unpopular option.

4) Basic intro to a procedural language. This extension is attractive if the class focuses on emphasizing the message that the skills attained in Scratch are easily transferable to "real" programming languages. A number of our class sections were extended with Python, and while it is not as unpopular as spreadsheets, it does not come close to the popularity of website design.

5) More complex block coding: (Snap, Blocky, Minecraft education). One of our sections used SNAP! as an extension. It was a powerful tool to showcase more advanced programming features, but the time devoted to it (~1/7th of the total class time) was not enough for it to be truly popular.

6) Mobile App creators (MIT App Inventor). One of our sections used MIT App Inventor [7], which has been shown to strongly ground computational thinking and to increase learning satisfaction for non-STEM students [8]. It allows the users to get some hands-on experience designing a user interface in a relatively easy-to-understand Java-like environment, and then they can write the back-end software for the app in a Scratch-like block code. The great disadvantage is that while the environment works both with Apple and Android devices, the completed apps can only be downloaded to Android devices as stand-alone applications. In the class section we tried this 28 out of the 29 students were using Apple devices, so they did not get roped into the allure of creating

one's own app. The sole Android user cited App Inventor as their favorite part of the course, however.

II. Background

Prior motivation and perceived level of difficulty often impact student success in introductory programming courses [9], [10]. Apart from prior work that we have reported on [11] other authors have also reported that fostering motivation and enhancing student engagement can have significant impact on student success in early programming courses [9], [12], [13].

A key issue cited is also the complexity of the programming language which can take away from the task of learning programming concepts and computational thinking and focusing on problem solving and concepts before the teaching of a programming language can significantly improve student's overall success [14]. Towards reducing complexity, the use of a simpler language such as Python has been proposed and utilized by many. Several researchers have also demonstrated the efficacy of utilizing block-based and point-and-click interfaces for the teaching programming concepts [15]. These are often used in classes that are meant to serve as a pre-introductory programming experience, thus fitting at a CS0 level in a traditional CS programming sequence or program. The advantage of these approaches is in introducing students to computing and computational thinking without getting bogged down in the more complex details [16]. They can introduce a wider body of students to computation and CS and drive the motivation for students who then choose to pursue CS can carry forward to their future coursework.

Schindler and Muller [17] found that " The first contact with programming is crucial to keep students in the long run", so if the curriculum design or educational approach allows for it, creating an appropriate "first contact" experience will be valuable. Their work as well as past work we have reported on [18] demonstrates that creative approaches in the CS0 pre-introductory courses can help towards addressing the gender gap in CS.

Computational thinking is also a vital skill in our increasingly computerized age [3], [4], [6] and this type of course can be valuable to students in various majors regardless of whether they decide to pursue computer science or computer science-adjacent coursework in some capacity.

III. Course descriptions and comparisons

So far, we have offered our pre-introductory experiences three times, for a total of 79 students. The three classes shared the same Scratch-based core, but had slightly different target audiences, extensions, credits and delivery.

A. First iteration: 2023 January Term - Coding is Fun

This class was our first offering of the pre-introductory experience, co-taught by both of us. This was a 1 Carnegie unit elective class that was offered for interested students as an *additional* learning opportunity on top of their required 3 Carnegie unit January Term course. This structure had some positive and negative effects. The good thing was, that this class was taken exclusively by students who were interested in trying their hands in programming, as the class did not fulfill any requirements. The bad thing was that the class was too short, which was the most common feedback that we have received from the students. To combat the relatively limited time in the class, three senior students served as TAs in the course, and both them and the two faculty members were available to the students for consultation when they were working on their projects.

This class intended to meet only 4 times, once each Wednesday from 9:15am to 11:45am over January. The first class focused on a quick introduction to Scratch, and then we allowed the students through a survey to select how they wanted to proceed with the course: continue with Scratch, have a quick introduction to Python or have a quick introduction to Spreadsheets. The student feedback indicated no interest in Spreadsheets, but many said they wanted to have *both* more Scratch *and* some immersion to Python. Because of this, the second class focused on Scratch, and an extra two optional classes were added to cover Python. The third official class focused on Computational and Design thinking, while the last class was set aside for the students to demo their projects.

While the intention of this class was to attract first year students, most of the students who signed up were graduating seniors, since seniors have priority when registering for classes. On the one hand this was great feedback, indicating that interest for a class like this is broader than we initially thought, but on the other hand the class ended up not serving the very population we created it for.

B. Second iteration: 2023 Fall - CS 102 Digital Literacy

The second and third interactions of the class were both 3 full Carnegie units, based on the feedback received during the previous semester. The class in the fall is a curricular requirement for the Digital Studies minor, but it was re-imagined along the above pre-introductory curriculum.

The first month of the course focused on Scratch, the second month on website programming and design, and the final month on Python programming. Each section concluded with its own mini-project, giving the students freedom to express their creativity while practicing the material covered.

The class attracted students from three major categories:

1. 16 first years interested in one of the computational disciplines, especially Programming or Analytics, were placed in this course by the Pre-enrollment team of the university. Incoming students fill out a Pre-Enrollment Questionnaire describing their interests, and then their schedule is predetermined by a team of experts ensuring that the first semester is in alignment with their interest and educational background, ensuring a smooth transition into college life. As part of this process the computationally interested students with no programming experience were placed in the course
2. 4 digital studies minors are required to take this class as their technical introduction to the digital world. These students tended to be sophomores, but some juniors were among them too
3. 5 students with primary interest in other areas who wanted to explore programming.

C. Third iteration: 2024 January Term - Coding is Fun

This section was developed parallel to the 2023 Fall class (paradoxically, the deadline for the course proposal for the 2024 January course was months before the deadline to finalize the 2023 Fall course). Just like the fall class, this iteration was also 3 Carnegie units, but the condensed nature of January Term necessitated the course to have slightly different structure.

Like the first iteration of the course, this class was offered as a series of 2.5 hour long class periods, for a total of 15 periods spread around 4 weeks, with 4 class periods most weeks. During the first (truncated, 3-day long) week, the basics of Scratch were introduced, with a practice mini-project assigned for the weekend. The second week concluded Scratch in the first two sessions, then devoted a class session to computational and design thinking, leading to a series of student "midterm" presentations that Friday. The second half of the January term was a series of miscellaneous topics sampled, with 2 days devoted to Snap!, a day each for spreadsheets and website basics, concluding in yet another weekend project homework. The final week spent two days on App Inventor, a day of intensive project consultation / development, and the final day for final presentations. This version of the class replaced the three mini-projects of the Fall interaction with 2 homeworks and 2 projects, but the spirit remained the same.

One peculiarity of this class iteration came from its modality. In order to accommodate student needs, this class was offered in an online format, to ensure it is available for anyone. This had the negative side effect that 11 of the 30 students who took the class admitted that they had no interest in programming at all. These students were also identifiable by the intensity of their participation, the intricacy of their projects, and their feedback at the end of the course.

IV. Results: Data

A. Data Sources

We have collected data about the student experience in three different ways. First, there was an active dialogue with the students in all of the classes. During multiple iterations, we even invited the students to select how the class should proceed. In the first iteration they chose that they wanted both Python and Scratch, so we extended the class with extra sessions. In the third iteration the student had an even split among those who wanted Snap!, App Inventor or Website, so the class accommodated all of those interests in a more-breadth-than-depth fashion.

The second, and for the purposes of this article the key way of collecting data was a pre- and post-survey in each class. In the prequestionnaire we gathered the students' a priori attitudes to programming, alongside with their interest and experience in the area. The students were asked how much prior coding experience they have, how hard they think coding will be for them during the class, how important coding will be for them in their lives, and whether they want to work in the industry or not. In the post questionnaire we asked about the **change** of their attitudes regarding the difficulty of programming, the usefulness of programming, the likelihood of them using programming in their everyday life, and how they plan on pursuing their coding education (from no more coding all the way to majoring in computer science).

The third source of data is to observe what the students actually *did*. How many of them followed up on their expressed interest in programming by taking an actual class.

B. Jan term 2023

1) Change in the perception of difficulty

When it comes to the difficulty of programming, a slight majority found coding easier than expected, while most of the rest found programming to be just as hard as they expected it to be. There was only a small minority of ~13% who found programming a little harder than they expected, but nobody found it much harder than their expectations. Overall, we can conclude that this class ended up being positively surprised regarding the difficulty of coding (table I).

TABLE I. JAN TERM 2023 DIFFIFULTY PERCEPTION

	Somewhat harder than expected	***About as hard as expected***	***Somewhat easier than expected***	***A lot easier than expected***
Too hard for me	0.00%	0.00%	0.00%	0.00%
I can probably do it	13.33%	**33.33%**	**33.33%**	0.00%
I can surely do it	0.00%	6.67%	6.67%	6.67%

2) Change in the perception of utility

When it comes to the utility of programming, the results are even more positive. Fewer than 7% of the students found programming to be less useful than they expected, while 80% realized that programming is more useful than their expectations (table II).

TABLE II. JAN TERM 2023 UTILITY PERCEPTION

	*I think coding will be **somewhat less useful** for me than before the course.*	*I think coding will **be as useful** for me as I thought before the course.*	*I think coding will be **somewhat more useful** for me than before the course.*	*I think coding will be **a lot more useful** for me than before the course.*
I probably **won't use** it	0.00%	6.67%	6.67%	0.00%
I'll **probably** hav use it	6.67%	6.67%	**33.33%**	6.67%
I'll **definitely** have to use it	0.00%	0.00%	**20%**	13.33%

3) Planned followup

The key feature regarding the follow-up for this cohort is the fact that over 40% of them were seniors, and only 4 were freshman. So unlike with the other two cohorts, the opportunity of an organized follow-through was significantly more limited for these people. Even so, only ~13% of the cohort did not expect to follow through with any kind of CS education, although most of them were expecting to keep on studying programming on their own. Interestingly, out of the 40% of the students who said they will follow up either by taking a class or even trying to go for a minor or certificate, two thirds of them already started by taking a class (figure 2).

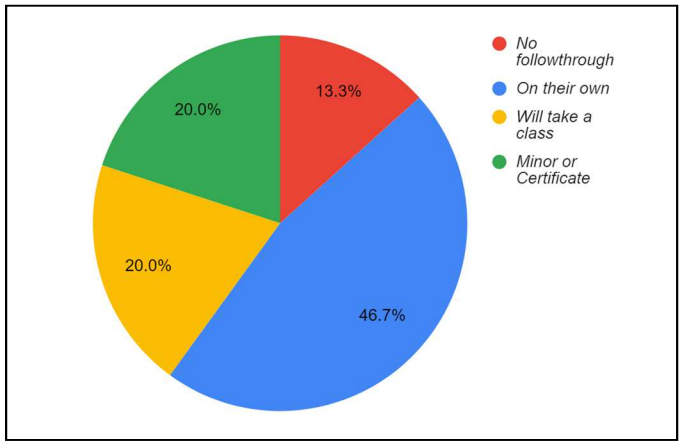

Fig. 2. Jan term 2023 planned follow-up

C. Fall 2023

1) Change in the perception of difficulty

When it comes to the difficulty of programming, we see a similar pattern as with the first class. A small minority, ~11%, found programming more difficult than expected, and about half of the remaining students found it easier than expected, while the rest found coding just as hard as expected (table III).

TABLE III. FALL 2023 DIFFICULTY PERCEPTION

	Somewhat harder than expected	***About as hard as expected***	***Somewhat easier than expected***	***A lot easier than expected***
Too hard for me	0.00%	0.00%	5.56%	0.00%
I can probably do it	11.11%	**38.89%**	**33.33%**	0.00%
I can surely do it	0.00%	5.56%	6.67%	5.56%

2) Change in the perception of utility

Regarding the utility of programming, this cohort felt that their expectations were met more closely than the students found in the first January Term. At the same time, a majority (56%) of them still found programming more useful than expected, and only 44% found it just as useful as expected (table IV).

TABLE IV. FALL 2023 UTILITY PERCEPTION

	*I think coding will be **somewhat less useful** for me than before the course.*	*I think coding will **be as useful** for me as I thought before the course.*	*I think coding will be **somewhat more useful** for me than before the course.*	*I think coding will be **a lot more useful** for me than before the course.*
I probably **won't use** it	0.00%	0.00%	5.56%	0.00%
I'll **probably** hav use it	0.00%	**33.33%**	**16.67%**	5.56%
I will **definitely** have to use it	0.00%	11.11%	**22.22%**	5.56%

3) Planned followup

Compared to the previous cohort, this cohort was relatively younger, so they had more time to follow up on their CS studies. This means we should expect them to want to follow up more -- and the data suggests just that. Only 5% of the responders (compared to over 13% in the previous class) said that they do not want to continue with programming, and the ratio of the people who wanted to continue in a "low quality" fashion (on their own or using online resources) dropped to 35% from 46%. 60% of respondents said they will want to take at least a class, but most of them wanted to get a certificate, minor or major. Encouragingly, even though only a single semester has passed since the class finished, 40% of the students already took or are enrolling to take an intro to programming course. Excitingly, even a student who said they will only study "on their own" is now enrolled in an actual intro class (figure 3).

D. Jan term 2024

This cohort of students was a curious group. Because the online modality of the class was very attractive to some, this class had a relatively large population of students actively disinterested in the class who simply took this because it was available online. Fundamentally, the student population here can be broken down into three groups:

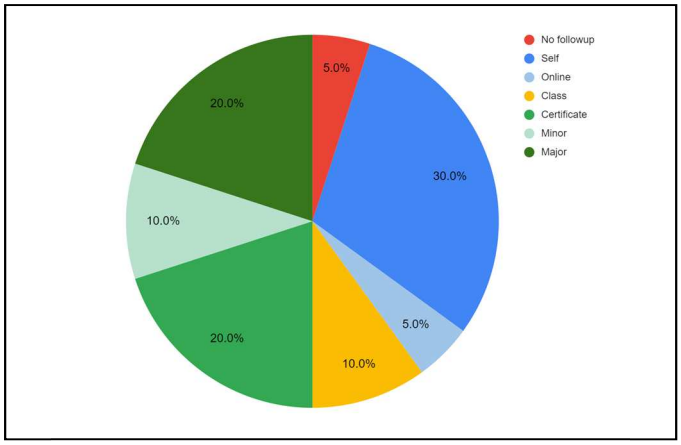

Fig. 3. Fall 2023 planned follow-up

- 11/29 students did not care about programming at all, only about the online modality of the course. These students' reported that they did not want to follow up, or only want to follow up through self-study.
- 6/29 students said that they did not care about the class being online, only about trying programming. This cohort exhibited the most robust response regarding follow-up intent: four reported wanting to major in the field, and one student said they were looking for a certificate.
- 12/29 students said they both cared about programming and the online modality as well. These students formed a spectrum regarding follow up, spread across all categories, but a majority still reported a desire for structured follow-up.
- One student did not fill out the survey.

Thus over a third of the students who were not interested in the material sadly took up room from a large number of students who wanted to join the class. Traditionally January Term courses are not allowed to support more than 20 students; we were allowed to offer more seats both semesters. At the same time, the 10-person waitlist of the class was full of students who wanted to add the course but could not. We are going to adjust the syllabus for the 2025 January Term to appeal more to the intended core audience and less to those who are only looking for an online experience. In order to keep the results comparable with the previous two classes, we report only on the 18 students who actually were interested to be in the class in the first place.

1) *Change in the perception of difficulty*

The perception of difficulty is very similar to the first two classes, with over 61% of the students reporting coding to be somewhat or a lot easier than expected, and nearly 28% reporting coding to be about as hard as they were expecting. Sadly, there was a student who found coding to be significantly harder than they expected, even though they thought it was something they could for sure do (table V).

TABLE V. JAN TERM 2024 DIFFICULTY PERCEPTION

	Somewhat harder than expected	***About as hard as expected***	***Somewhat easier than expected***	***A lot easier than expected***
Too hard for me	0.00%	0.00%	5.56%	0.00%
I can probably do it	0.00%	0.00%	0.00%	0.00%
I can surely do it	11.11%	**27.78%**	**33.33%**	**22.22%**

2) *Change in the perception of utility*

Once again, a majority (61%) is reporting that after the class they feel that programming will be somewhat or a lot more useful than they expected. Sadly, the student who expressed difficulty with programming is also finding it to possibly be much less useful in the future. Still, an overwhelming majority of students report encouraging results (table VI).

TABLE VI. FALL 2023 UTILITY PERCEPTION

	*I think coding will be **somewhat less useful** for me than before the course.*	*I think coding will **be as useful** for me as I thought before the course.*	*I think coding will be **somewhat more useful** for me than before the course.*	*I think coding will be **a lot more useful** for me than before the course.*
I probably **won't use** it	0.00%	0.00%	5.56%	0.00%
I'll **probably** hav use it	11.11%	**16.67%**	**22.22%**	11.11%
I'll **definitely** have to use it	5.56%	5.56%	11.11%	11.11%

3) *Planned followup*

Among the interested students, the desire to follow up is even more prominent than the previous two cohorts. While about 11% of the students do not want to further their programming education, the "low quality" follow-ups of "self" and "online" are only about 28%, and 56% want to pursue at least a certificate, if not a minor or a major. And even though only a single semester passed since this class, already 33% of the students took a follow-up introductory course (figure 4).

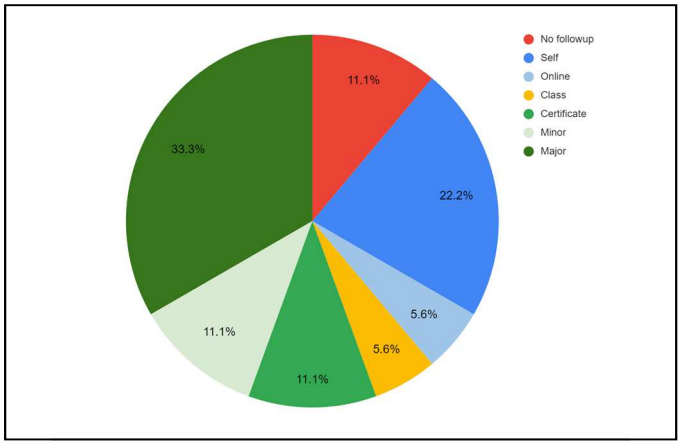

Fig. 4. Jan term 2024 planned follow-up

V. RESULTS: OUTCOMES AND DISCUSSION

Two years ago we have set out to help our potential students overcome their fear of programming by providing them with a set of pre-introductory courses aimed at reducing the perceived difficulty and increasing the perceived utility of the discipline. After three sections of classes and nearly 80 students taught, we have found that students who participate in a pre-introductory class tend to find programming easier and more useful than expected.

A. *Perceived difficuly summary*

Overall about 51% of students find coding easier than expected, 37% of students find it as hard as expected and 12% of students find it more difficult than expected. While this 12% figure can be disheartening, in the context of the reality that **most of the students in these classes would not have even tried programming** if all they had available was an introduction to programming course, these results are actually quite encouraging (table VII).

TABLE VII. PERCEIVED DIFFICULTY SUMMARY

	Somewhat harder than expected	***About as hard as expected***	***Somewhat easier than expected***	***A lot easier than expected***
Too hard for me	0.00%	0.00%	3.92%	0.00%
I can probably do it	7.84%	**23.53%**	**21.57%**	0.00%
I can surely do it	3.92%	13.73%	13.73%	11.76%

1) *Perceived utility summary*

Once again, a majority (61%) is reporting that after the class they feel that programming will be somewhat or a lot more useful than they expected. Sadly, the student who expressed difficulty with programming is also finding it to possibly be much less useful in the future. Still, statistically a vast majority of students report encouraging results, with easier and more useful than expected programming (table VIII).

TABLE VIII. PERCEIVED UTILITY SUMMARY

	*I think coding will be **somewhat less useful** for me than before the course.*	*I think coding will **be as useful** for me as I thought before the course.*	*I think coding will be **somewhat more useful** for me than before the course.*	*I think coding will be **a lot more useful** for me than before the course.*
I probably **won't use** it	0.00%	1.96%	5.88%	0.00%
I'll **probably** hav use it	5.88%	**19.65%**	**23.53%**	7.84%
I'll **definitely** have to use it	1.96%	5.88%	17.65%	9.80%

B. *Planned Follow-up*

Fewer than 10% of the students remained unconvinced after the course that programming is something that they should study further. Again, in the context that a majority of these students would never have even tried out programming, this result is encouraging. A further 35% plan to study programming on their own. While this was labeled as "low quality" follow up in the above passages, the reality is that with today's technology, through the help of the large amounts of online resources, support websites, videos, simulation environments, it is quite plausible for someone to teach themselves quite a significant amount of programming on their own. A further 12% plans to at least continue on to an intro class, and more than 40% want to at least secure a certificate in the field. Reinforcing these strong numbers, already nearly 31% of the students in the three pre-introductory programming classes took an introductory course, so it is likely that by the time they graduate we will get close to this ~ 50% mark.

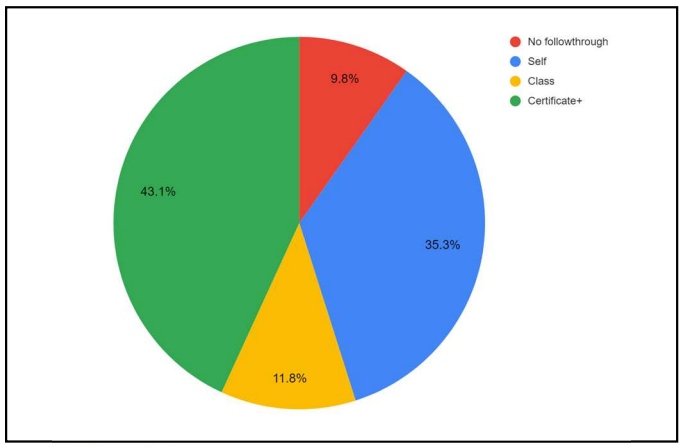

Fig. 5. Planned follow-up summary.

VI. LIMITATIONS OF THE APPROACH

While we are excited by the results that this approach works successfully in our setting for our target audience, there are a number of considerations that need to be mentioned.

Our positive results were among students who self-selected into our elective courses. This presupposes a level of personal interest on their part; something that might not be present in a mandatory class. This, of course, highlights one of the benefits of such a pre-introductory class: it provides an opportunity for students who are programming curious but might be too afraid to take an intro to programming class to try their hand at programming. Our approach can be considered as an alternative ramp into the realm of programming, not a one-size-fits-all solution to all of the issues with the introductory programming experience. This fact is illustrated by the reality that throughout the most recent iteration of the class, none of the students who were initially disinterested in programming and only took the course because of its modality reported that they gained significant interest in programming -- despite the fact that a number of them were really talented and successful.

VII. CONCULSIONS AND NEXT STEPS

The success of this innovative approach is evident from the above results. We plan on continuing offering the classes both in the Fall semester and over January term, trying to encourage students to brave a programming class. The intent is not to turn everybody into a programmer, but rather to allow as many students as possible to infuse their natural passion, whether it is in the arts or sciences or business studies with some programming, to empower them to be more successful in their chosen field of expertise. If along the way some choose computer science, then that's a lucky bonus.

As seen in the planned follow-up summary, at least half the students are opting to continue study in computer science or data science which is a strong signal for continuing to offer and further develop these courses. Our concrete data of which students have chosen a CS/DS major, minor, or certificate, are also encouraging and we hope to be able to report on what the students who said they would "follow-up" with CS/DS actually ended up doing in the long run.

REFERENCES

[1] U. Das and C. Fulton, "Reducing Barriers to Entry by Removing Prerequisites for a CS1 Introductory Programming Course," in *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 2*, in SIGCSE 2024. New York, NY, USA: Association for Computing Machinery, Mar. 2024, pp. 1616–1617. doi: 10.1145/3626253.3635492.

[2] "Snap! Build Your Own Blocks." Accessed: May 12, 2024. [Online]. Available: https://snap.berkeley.edu/

[3] "Computational thinking is critical thinking: Connecting to university discourse, goals, and learning outcomes - Kules - 2016 - Proceedings of the Association for Information Science and Technology - Wiley Online Library." Accessed: May 19, 2024. [Online]. Available: https://asistdl.onlinelibrary.wiley.com/doi/10.1002/pra2.2016.14505301092

[4] A. E. Weinberg, "Computational Thinking: An Investigation Of The Existing Scholarship And Research," PhD Thesis, Colorado State University, 2013.

[5] J. Moore, J. Sanchez, and A. Tudor, "Weaving a Storytelling Tapestry Using Computational Thinking," *IASL Annual Conference Proceedings*, Sep. 2021, doi: 10.29173/iasl8306.

[6] B. H. de Paula, A. Burn, R. Noss, and J. A. Valente, "Playing Beowulf: Bridging computational thinking, arts and literature through game-making," *International Journal of Child-Computer Interaction*, vol. 16, pp. 39–46, Jun. 2018, doi: 10.1016/j.ijcci.2017.11.003.

[7] "MIT App Inventor." Accessed: May 19, 2024. [Online]. Available: https://appinventor.mit.edu/

[8] C. H. Liao, C.-T. Chiang, I.-C. Chen, and K. R. Parker, "Exploring the relationship between computational thinking and learning satisfaction for non-STEM college students," *International Journal of Educational*

Technology in Higher Education, vol. 19, no. 1, p. 43, Aug. 2022, doi: 10.1186/s41239-022-00347-5.

[9] O. Solarte Pabón and L. Machuca Villegas, "Fostering Motivation and Improving Student Performance in an introductory programming course: An Integrated Teaching Approach," *reveia*, vol. 16, no. 31, pp. 65–76, Jan. 2019, doi: 10.24050/reia.v16i31.1230.

[10] A. Gomes and A. Mendes, "A teacher's view about introductory programming teaching and learning: Difficulties, strategies and motivations," in *2014 IEEE Frontiers in Education Conference (FIE) Proceedings*, Oct. 2014, pp. 1–8. doi: 10.1109/FIE.2014.7044086.

[11] A. Margitay Becht and U. Das, "Enhancing student learning through hidden motivational learn-ing outcomes.," in *Enhancing student learning outcomes in higher education.*, Libri Publishing Ltd., 2023.

[12] B. Pérez and Á. L. Rubio, "A Project-Based Learning Approach for Enhancing Learning Skills and Motivation in Software Engineering," in *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, in SIGCSE '20. New York, NY, USA: Association for Computing Machinery, Feb. 2020, pp. 309–315. doi: 10.1145/3328778.3366891.

[13] R. P. Medeiros, G. L. Ramalho, and T. P. Falcão, "A Systematic Literature Review on Teaching and Learning Introductory Programming in Higher Education," *IEEE Trans. on Educ.*, vol. 62, no. 2, pp. 77–90, May 2019, doi: 10.1109/TE.2018.2864133.

[14] T. Koulouri, S. Lauria, and R. D. Macredie, "Teaching Introductory Programming: A Quantitative Evaluation of Different Approaches," *ACM Trans. Comput. Educ.*, vol. 14, no. 4, p. 26:1-26:28, Dec. 2015, doi: 10.1145/2662412.

[15] M. E. Caspersen, "Educating Novices in the Skills of Programming," PhD Thesis, University of Arhaus.

[16] A. Funke, K. Geldreich, and P. Hubwieser, "Analysis of scratch projects of an introductory programming course for primary school students," in *2017 IEEE Global Engineering Education Conference (EDUCON)*, Athens, Greece: IEEE, Apr. 2017, pp. 1229–1236. doi: 10.1109/EDUCON.2017.7943005.

[17] C. Schindler and M. Müller, "Gender gap? a snapshot of a bachelor computer science course at Graz University of Technology," in *Proceedings of the 13th European Conference on Software Architecture - Volume 2*, in ECSA '19. New York, NY, USA: Association for Computing Machinery, Sep. 2019, pp. 100–104. doi: 10.1145/3344948.3344969.

[18] A. Margitay Becht and U. Das, "Supporting Gender Equality in Computer Science Through Pre-Introductory Programming Courses", doi: 10.5281/zenodo.8431898.