Good Students are Good Students
Student Achievement with Visual versus Textual Programming

Joel Coffman, Justin M. Hill, Shannon Beck, Adrian A. de Freitas, Troy Weingart
Department of Computer and Cyber Sciences, United States Air Force Academy
Email: { joel.coffman, justin.hill, shannon.beck, adrian.defreitas, troy.weingart }@usafa.edu

***Abstract*—In this full research paper, we compare the impact of learning a visual versus textual programming language in an introductory computing course that is a general education requirement at our institution. We conducted a randomized comparative study with "experimental" sections that were taught using Python instead of RAPTOR, a flowchart-based programming language. The populations of students learning each programming language were similar with respect to gender, race, and predicted performance based upon standardized test scores and prior post-secondary education. Although students' performance on the whole was similar regardless of the programming language taught, predicted performance is correlated with SAT Math scores, grades in mathematics courses (specifically Calculus II), and, for lower-performing students, grades in other courses that satisfy general education requirements. That is, students from these groups who had lower predicted performance and learned Python performed worse on average than their peers who learned RAPTOR, and students with higher predicted performance outperformed (on average) their peers who learned RAPTOR. In addition, students' performance in subsequent computer science courses was not correlated with their performance and the language they learned in our introductory computing course. Our results raise important questions about the role of an introductory computing course in promoting equity and engaging students from historically underrepresented groups in computing fields.**

***Index Terms*—computer science, undergraduate, instructional change, achievement, equity, introduction to programming, RAPTOR, Python**

I. Introduction

Opinions regarding the language to use in introductory programming courses have varied widely for decades [26], particularly following the replacement of Pascal as the *lingua franca* across computer science curricula. Quoting from the 2013 ACM / IEEE Computer Society computer science curriculum guidelines [25]:

> ...rather than a particular paradigm or language coming to be favored over time, the past decade has only broadened the list of programming languages now successfully used in introductory courses. There does, however, appear to be a growing trend toward "safer" or more managed languages (for example, moving from C to Java) as well as the use of more dynamic languages, such as Python or JavaScript. Visual programming languages, such as Alice and Scratch, have also become popular choices for providing a "syntax-light" introduction to programming; these are often (although not exclusively) used with non-majors or at the start of an introductory course.

The ongoing debate regarding the difficulty [21] or ease [32] of learning to program indicates that this issue remains relevant. The majority of students in an introductory computing course that fulfills a general education requirement will not go on to major in a computing-related discipline. For many, such a course may be their only formal exposure to programming. Unfortunately, existing research is mixed regarding which programming language (or, more generally, programming paradigm) is most appropriate in this context.

The 2019–2020 academic year presented a unique opportunity to study the impact of changing the programming language from RAPTOR [10], a flowchart-based programming language, to Python in our introductory computing course. Unlike many other institutions, our introductory computing course is required for all students, allowing us to examine the impact of this change from a variety of perspectives due to the lack of self-selection bias [23]. More specifically, we conducted a randomized comparative study by randomly assigning students to sections taught using either RAPTOR or Python.

In this paper, we examine the relationship between the programming language used in an introductory computing course (i.e., RAPTOR or Python) and predicted performance, both in the introductory course and in subsequent technical courses. Our contributions are as follows:

- We explore factors that predict student performance in our introductory computing course, finding a stronger relationship between mathematical maturity and achievement when learning a textual programming language.
- We analyze how students' initial programming language affects their performance in subsequent courses, finding that the differences are not statistically significant.

Our work is part of a comprehensive effort to better understand the impact of changing the programming paradigm used to teach introductory programming.

The remainder of this paper is organized as follows. We present an overview of our research methodology including our research questions, experimental design, participants, and data collection in Section II. Section III details our evaluation of the change from RAPTOR to Python. We discuss our results in Section IV. In Section V, we summarize related work. Finally, we conclude in Section VI.

II. Research Method

We performed a randomized comparative experiment to evaluate the impact of the programming language in an

introductory computing course that is required for all students at our institution. Randomization is the gold standard when attempting to show a causal relationship between different treatments [13], [36] (e.g., the programming language).

A. Background

Comp Sci 110: Introduction to Computing and Cyber Operations is a general education requirement at the United States Air Force Academy and typically taken by students during their freshman year. Comp Sci 110 is best characterized as CS0.5 [41]: it introduces computer science principles (e.g., binary), exposes students to programming, and surveys related fields (e.g., artificial intelligence (AI) and cybersecurity). Programming comprises approximately 50% of the course material. Although some students have taken prior computer science courses, either in high school (e.g., AP courses) or at other colleges, few students receive placement or transfer credit due to the non-programming topics covered.

This 3-credit course has one lecture hour and two hours of activities outside of each class session. Activities outside of class are standardized across sections—i.e., everyone has the same readings, homework, programming exercises, etc. While instructors have flexibility regarding their use of class time, most split the period into "lecture" and "lab" time. Instructors review prior material and cover new material at the beginning of the class, and students use the remaining time to work on homework while their instructor can provide immediate assistance.

During the 2019–2020 academic year, we switched the programming language used for instruction in Comp Sci 110 from RAPTOR [10] to Python. The decision to change the programming language was driven by several factors, including perceived limitations of visual programming (e.g., difficulty following established software engineering practices such as avoiding global variables [51], functional decomposition [37], and unit testing) and perceived advantages to introducing students to a programming language that they may use later (e.g., in subsequent courses such as AI, in related majors such as cybersecurity and data science, and after graduation). In addition, Python is used by an increasing number of introductory programming courses [7]. In this regard, we are following a broader trend of adopting a programming language that is already in common use in industry [22].

B. Research Questions

We examine the relationship between student achievement and the use of a visual or textual programming language in an introductory computing course. Student achievement includes performance in the introductory computing course but also encompasses subsequent courses that build upon these concepts across a variety of degree programs.

We address the following research questions:

1) **Predicting Performance** What factors (e.g., standardized test scores and mathematical maturity) predict student performance in an introductory computing course?
2) **Subsequent Performance** Does the programming language used in an introductory computing course affect students' achievement in subsequent technical courses, including programming courses?

TABLE I
STATISTICS FOR COURSE OFFERINGS

	RAPTOR		Python	
Semester	Sections	Enrollment	Sections	Enrollment
Fall 2019	22	515	4	94
Spring 2020	14	301	9	173
Overall	36	816	13	267

C. Experimental Design

A unique aspect of our work comes from Comp Sci 110 being taken by all students at the United States Air Force Academy. There is no possibility of self-selection bias [23]. Having a single course differs from efforts that contextualize computing based on students' majors (e.g. STEM vs. liberal arts) [20] or segregating sections based on prior experience [11]. Thus, we can analyze the impact of changing the programming language across an entire population of first-year undergraduates.

In Fall 2019, we taught several experimental sections using Python. The Python sections had the same schedule, learning outcomes, etc. as RAPTOR sections to enable an unbiased comparison. Instructors collaborated closely to ensure assignments and assessments were worded similarly and independent of the programming language. An interactive textbook was used for Python with the premise it would increase student engagement (see Edgcomb et al. [16]), potentially leveling the learning curve of a text-based language.

Due to the initial success with Python, the number of experimental sections increased in Spring 2020. That semester we relaxed the policy of keeping the RAPTOR and Python versions of the course in lockstep so that the experimental sections could better align with existing resources (e.g., the textbook) while maintaining the same learning outcomes.

D. Participants

Our participants include all students who took Comp Sci 110 in the 2019–2020 academic year (i.e., the entire Class of 2024 with a handful of exceptions such as students required to retake the course). In total, 1083 students participated in our study (see Table I) with 25% learning Python.

Sampling: The university registrar randomly assigned students to sections. Students had equal chance of being assigned to a section using RAPTOR as to a section using Python based on their schedule. An experimental section was not offered every period, which accounts for the different proportion of students who learned each programming language. Transfers between sections using different programming languages were not permitted even when requested by students and feasible due to section sizes and students' class schedules. Anecdotally, many students, particularly those who had prior programming

TABLE II
STUDENT DEMOGRAPHICS (% OF STUDENTS). DATA WAS NOT AVAILABLE FOR 116 STUDENTS (10.7%); THE PERCENTAGES OMIT THESE STUDENTS.

	Gender		Race					
Language	Female	Male	Asian	African American	Caucasian	Hispanic	Other	Unknown
RAPTOR	28.2	71.8	10.6	10.2	65.4	10.1	1.8	2.0
Python	28.3	71.7	8.8	10.0	65.7	11.2	2.0	2.4
Overall	28.0	72.0	9.8	10.2	65.9	10.2	1.9	2.1

TABLE III
STANDARDIZED TEST SCORES AND GPA IN COURSES THAT SATISFY GENERAL EDUCATION REQUIREMENTS.

	SAT		ACT		
Language	Math	Combined	Math	Composite	GPA
RAPTOR	678	1338	29	29	2.978
Python	678	1337	29	29	3.002
Overall	678	1337	29	29	2.987

experience, asked to switch from a RAPTOR section to a Python section whereas we did not observe a similar number of requests by students learning Python.

For a randomized comparative experiment, the groups of subjects (i.e., students learning each programming language) should be similar on average to mitigate the risk of confounding variables. Tables I, II, and III provide statistics regarding the course enrollment, demographic data, and standardized test scores for our students. In Table II, "Other" combines the small number of Native American and Hawaiian / Pacific Islanders to avoid identification of individuals; "Unknown" indicates students did not provide this data. The distribution of students learning each programming language is similar. Students' standardized test scores (ACT and SAT) and grade point average (GPA) in courses that fulfill general education requirements were also comparable (see Table III). Consequently, we conclude that the assignment of students to learn each programming language was effectively randomized and any differences in students' performance result from the programming language being taught.

E. Data Collection

The data used in this study is archival, available from our registrar's student information system (e.g., final grades in courses taught by other departments) or our learning management system (LMS). All sections taught using the same programming language shared assignments and assessments. We used standardized rubrics to minimize scoring differences across instructors.

III. EVALUATION

This section details our evaluation, specifically the analysis of the data that addresses our research questions. We start by summarizing students' performance with and perception of each programming language.

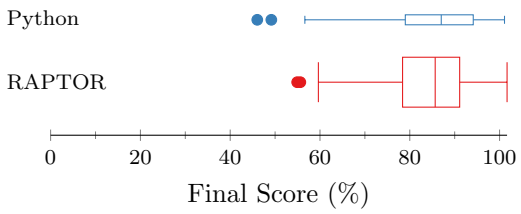

Fig. 1. Comparison of student performance using RAPTOR and Python.

Students across all sections of Comp Sci 110 performed similarly (see Figure 1). More specifically, the programming language did not have a statistically-significant difference in students' final scores (using a two-sided Welch's unequal variances t-test [50]). Interesting, the term in which the course was taken (Fall 2019 or Spring 2020) *did* have a significant effect, possibly due to the inclusion of more advanced programming topics (e.g., file I/O and two-dimensional lists) in Spring 2020 for students who learned Python or due to disruptions caused by COVID-19.

When comparing across demographic groups, there was not a statistically-significant difference by gender, but one racial group, African Americans, performed worse when learning Python ($p < 0.05$). Our current work, specifically examining factors that predict performance in our introductory computing course, is an attempt to understand this difference. For example, academic background (e.g., lack of computing courses or college-level mathematics courses available to a student while in high school) may account for this decrease, as we did not explicitly control for students' backgrounds when assigning them to a specific programming language. Additional analysis revealed that African American students who learned RAPTOR scored, on average, 60 points higher on SAT Math and earned a 0.103 higher GPA in courses satisfying general education requirements than African American students who learned Python; similar differences exist for other demographic groups.

Students' perception of each programming language varied widely. In a nutshell, students' overwhelmingly preferred to learn a programming language that they could use in the future (i.e., Python). Quoting one:

> I think [learning Python] would have not only been more interesting but in general it would've been a better use of my time since Python is widely applicable to the real world. I didn't really gain much out of Raptor, after the final I'll probably never use it again.

Despite the preference for learning Python, the number of computer science and cybersecurity majors did not change significantly, and the proportion of declared majors who learned RAPTOR and Python was consistent with the average overall enrollment (see Table I).

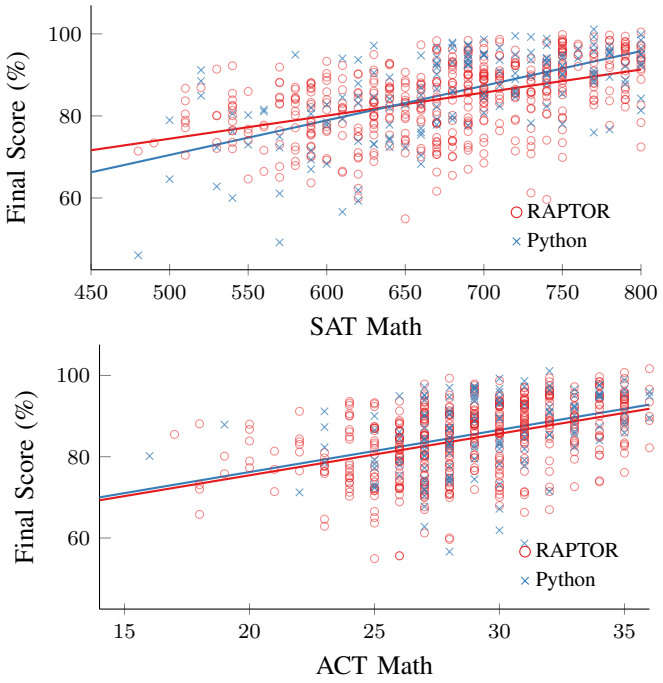

Fig. 2. Predicted student performance by standardized test score—SAT Math (top) and ACT Math (bottom).

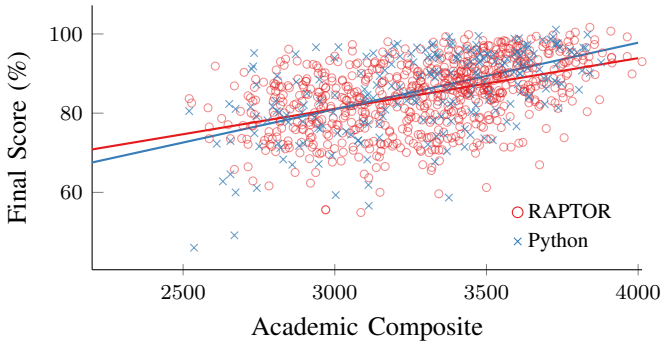

Fig. 3. Predicted student performance by academic composite.

A. *Predicting Student Performance*

Our first research question focuses on identifying underlying factors that predict student success in an introductory programming course. Knowing these factors facilitates targeted intervention for at-risk students and has been a topic of interest for decades (e.g., [4], [31]). From prior studies, mathematical maturity, as measured in various forms such as standardized test scores [8], [29], [39] and recent coursework [29], [40], are most often correlated with success. Unfortunately, few models have ever been revalidated, either longitudinally or across institutions although Quille and Bergin's work [39] is a notable exception to this trend.

Figure 2 contains a scatter plot and linear regressions for standardized tests and students' final score in our introductory course with each programming language. A robust linear regression indicates that a student's SAT Math score, the programming language, and interaction between these two variables predict a student's Comp Sci 110 score; each of these terms is statistically significant ($p < 0.005$). Conversely, there is not a statistically-significant relationship between a student's ACT Math score and the programming language in Comp Sci 110. A higher ACT Math score does predict better performance; i.e. "good" students perform better independent of the programming language used.

Our institution also has an internal estimate for students' academic performance. This "academic composite" score incorporates a variety of factors, including standardized test scores, strength of high school curriculum, and prior performance of similar students. Figure 3 reinforces the trend in Figure 2. A robust linear regression indicates that a student's academic composite and the interaction between the academic composite and programming language being used are statistically significant ($p < 0.02$ for each term in the model). This result suggests that teaching RAPTOR is more equitable than teaching Python, providing more uniform achievement across students with different levels of preparation for post-secondary coursework, although the magnitude of the effect is fairly small. More specifically, students in the RAPTOR course who were near the bottom of the range for academic composite scored, on average, 2.4% higher than their peers in the Python course whereas students in the RAPTOR course near the top of the range scored 4.2% lower than those in the Python course. The improvement in students' scores at the bottom of the range mirror prior research where lower-performing students using RAPTOR outperformed their peers who used other programming tools [9].

Figure 4 compares students' performance in Calculus I and II (taken at our institution) with Comp Sci 110. These mathematics courses may be taken before, after or concurrently with Comp Sci 110, depending on students' performance on placement exams and the semester in which they take Comp Sci 110 (i.e., fall or spring). The paired box plots in Figure 4 reinforce the prior theme: students who do well in Calculus I and II generally do better in Comp Sci 110. A two-way analysis of variance (ANOVA) indicates that the student's grade in Calculus I has a statistically significant effect ($p < 1e{-}40$) on performance in Comp Sci 110. However, the effect of the interaction between the programming language and Calculus I grade is not as significant ($p < 0.1$). Nevertheless, we see a trend similar to the one we observed with SAT Math scores: students in the RAPTOR course with the lowest Calculus I grades scored, on average, 4.6% higher than their peers in the Python course, whereas students in the RAPTOR course with the highest Calculus I grades scored 1% lower. It seems again that RAPTOR "levels the playing field" between students. The effect is more pronounced when considering instead students' grades in Calculus II. In this case, a student's grade in Calculus II also has a statistically significant effect ($p < 1e{-}42$) on performance in Comp Sci 110, but the effect of the interaction between the programming language and Calculus II grade is more significant than with Calculus I ($p < 0.03$). Moreover,

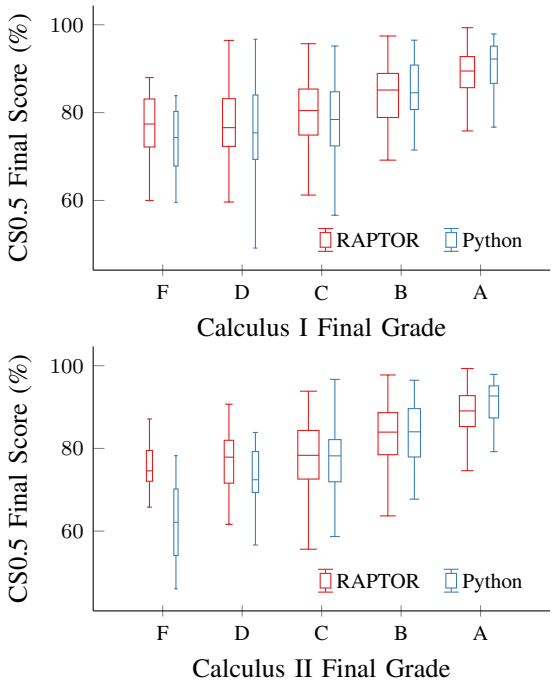

Fig. 4. Student performance in mathematics courses—Calculus I (top) and Calculus II (bottom) compared to our introductory computing course (y axis of both graphs). These courses are typically all taken as a freshman.

the equalizing effect of RAPTOR is even more substantial. Students in the RAPTOR course with the lowest Calculus II grades scored, on average, 16.6% higher than their peers in the Python course, whereas students in the RAPTOR course with the highest Calculus II grades scored 2.6% lower. As indicated by the widths in the provided box plots, some of these groups are small (e.g., the number of students in the Python course that earned an F in Calculus II). Nevertheless, we observe this "equalizing" effect for students learning RAPTOR when predicting student performance in several ways: by SAT Math and academic composite (as evidenced by the steeper slope for the Python regression line) and by Calculus I and II (by the steeper *implied* slope if one was to "connect the dots" between the Python box plots).

In light of the prior analysis, we were curious if the programming language specifically disadvantaged students with lower standardized test scores and GPAs in courses that satisfy general education requirements.[1] Using a two-sided Welch's unequal variances t-test ($\alpha = 0.05$), there was not a statistically-significant difference between students who scored in the bottom quartile of their peers at our institution with respect to SAT Math scores, but there was a statistically-significant difference between students with GPAs in the bottom quartile.

B. Subsequent Course Performance

Our second research question is how the programming language affects students' performance in subsequent computing courses. Although we recognize that most students in Comp Sci 110 will not major in a computing discipline (about 5% of our students major in computer science or cybersecurity any given year), most student have not previously taken another computer science or programming course. Thus, we also want to give students a realistic and authentic impression of the major, particularly because subsequent courses are taught using a textual programming paradigm. In fact, we teach C in our CS1 and CS2 sequence, and it is not obvious that either RAPTOR or Python is ideally suited to preparing students to program in a statically-typed language that requires explicit memory management.

[1]We specifically focus on GPA for general education requirements isolated from overall GPA to account for differences across major-specific courses.

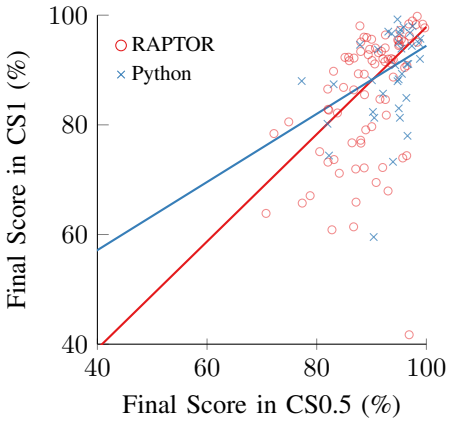

Fig. 5. Comparison of performance in CS1 by initial programming language

Figure 5 compares students' performance in CS1 based on the programming language learned in Comp Sci 110. Once again, students' performance in CS1 is correlated with their performance in our introductory computing course. A robust linear regression indicates that the programming language in Comp Sci 110 was not a significant predictor of the student's CS1 score. In fact, the only significant term in the model was the score in our introductory computing course ($p < 0.05$). Consequently, we reject the null hypothesis that learning a textual programming language better prepares students for our CS1 course.

Our department also offers several programming courses designed for engineering majors. These courses may be loosely characterized by the CS0.75 moniker insofar as they cover less material than a traditional CS1 course that is designed for students majoring in computer science. These courses use either MATLAB or Python and range from 1–3 credit hours; in addition, each course starts by reviewing the programming topics covered in Comp Sci 110 (variables, decisions, iteration, etc.). Like our CS1 course, a linear regression indicates that the programming language does not have a statistically-significant impact on students' scores in these subsequent programming courses designed for non-computer science majors (see Figure 6).

We also considered performance in two subsequent courses that use programming: Calculus III and Systems Analysis, the latter of which exposes students to quantitative modeling

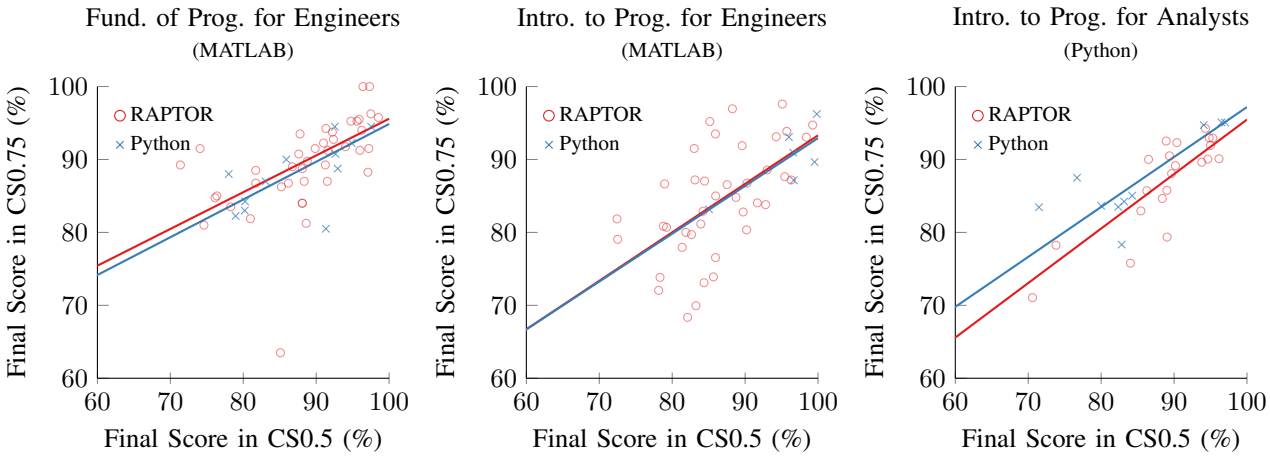

Fig. 6. Comparison of performance in CS0.75 courses by initial programming language. All axes span the same values to facilitate comparison, but we do not intend to extrapolate beyond the data (which is typically clustered in the range 80–100).

methods. In both cases, a two-way ANOVA indicates that neither the programming language nor the interaction between programming language and score in Comp Sci 110 is statistically significant. In other words, students' performance in subsequent courses that use programming (but do not require students to write programs from scratch) is not affected by the programming language they learned in Comp Sci 110.

To summarize, learning Python in an introductory computing course neither advantages nor disadvantages students in subsequent courses. This result is consistent for subsequent programming courses and technical (e.g., mathematics) courses.

IV. Discussion

Our analysis of SAT Math scores suggests learning a visual programming language (i.e., RAPTOR) advantages students who are less prepared whereas a textual programming language (i.e., Python) advantages students who are more prepared (see Figure 2). In contrast, ACT Math scores do not predict students' performance in Comp Sci 110. Most students take either the SAT or the ACT (but not both), and the SAT Math prediction is strongly influenced by several outliers—i.e., students who had a low SAT Math score and learned Python. Consequently, and despite the statistical significance of the terms in the SAT Math regression, this result might be spurious because we do not observe a similar correlation with the ACT Math standardized test scores.

Comparing performance in recent mathematics courses is also mixed: there is not a statistically-significant difference for Calculus I ($\alpha = 0.05$), but there is a statistically-significant difference for Calculus II. Moreover, the impact to students' scores increases with lower grades in Calculus II. Students who learned Python and failed Calculus II had a mean decrease of 16.6 points in Comp Sci 110 compared to students who learned RAPTOR. Comparing the bottom quartile of students based on GPA in general education courses again results in a statistically-significant difference between students learning each programming language. One conclusion from the analysis is that textual programming languages (e.g., Python) perpetuate existing differences among students whereas visual programming languages (e.g., RAPTOR) are more equitable because performance is less strongly predicted by students' backgrounds.

We expected teaching Python in an introductory computing course to benefit students who go on to take CS1. Our CS1 course is taught using C, and we speculated that Python would provide a more realistic perspective of programming activities than RAPTOR. At the very least, Python forces students to grapple with language syntax whereas RAPTOR glosses over these details, focusing instead on "algorithmic thinking" and problem solving. In reality, learning a visual versus textual programming language had no effect in subsequent courses, even those designed for computer science majors.

Our results raise several interesting questions regarding the objective of an introductory computing course. If the objective is to expose students to programming, then using a visual programming paradigm may increase equity. If the objective is to expose students to a programming language that they might use in the future, then Python is almost certainly a better choice given its vibrant ecosystem and syntactic similarity with other widely-used programming languages. Given what we know about the importance of interest to students' motivation [15] and how students' interests influence programming language preferences [14], one conclusion is that a "one size fits all" approach is inherently inferior to contextualized computing [20], both in promoting equity and in capturing students' interest in computing.

Threats to Validity

We discuss a number of potential threats to the validity of our results in the remainder of this section.

a) Subject Characteristics: Our study participants (i.e., students enrolled in Comp Sci 110) were randomly assigned to sections, which controls for selection bias [5] and increases internal validity [6]. Students learning RAPTOR and Python

had similar gender and racial distributions (Table II) and their academic performance is also similar (Table III). Thus, any differences in students' performance are attributable to the programming language taught in Comp Sci 110.

b) Differences among courses: In Fall 2019, the RAPTOR and Python courses had the same schedule, learning outcomes, etc. Instructors collaborated on assignments and assessments to ensure that they were comparable and independent of the programming language. To minimize differences between the RAPTOR and Python environments, we used Thonny [3], a Python integrated development environment (IDE). Thus, the most significant difference was the use of an interactive textbook for students learning Python, but recent studies [1], [30] indicate that reading comprehension is comparable regardless of the medium, either print or digital. In Spring 2020, we relaxed the policy of keeping both versions of the course in lockstep, but shared the same three learning outcomes, which were assessed similarly. Regardless of the semester, all sections using the same language had the same schedule, learning outcomes, etc.

c) Differences among instructors: We had a large number of instructors: thirteen for RAPTOR and nine for Python. Only four instructors taught sections in both RAPTOR and Python, and only two instructors taught both programming languages in the same semester. While we do not discount that some instructors may be more effective than others, most have significant prior teaching experience, and all instructors using the same programming language followed the same schedule with identical learning objectives and assessments.

d) Instrumentation Threat: This threat arises from instruments used in the study having low reliability, which can occur in a number of ways, including differences among assessment mechanisms and inconsistent scoring by an instructor (intra-rater reliability) or across multiple instructors (inter-rater reliability). To minimize this threat in the Fall 2019 semester, assessments were essentially identical including the grading rubrics for RAPTOR and Python. As with the potential for differences among instructors, comparing students' performance for instructors who taught both RAPTOR and Python is warranted as part of future work.

e) Generalization: Like most studies, our results may not generalize to students at other institutions. In particular, admission to the United States Air Force Academy is highly competitive, ranking among the most selective colleges and universities in the country. Given the admissions process and rigor of the curriculum, our students may be better prepared for an introductory computing course that includes a substantial amount of programming, which may minimize the the benefit of using a visual programming paradigm, particularly for groups that are historically underrepresented in computing [46].

V. Related Work

There is surprisingly little empirical data to support the choice of which programming language(s) to use in introductory programming courses. Pears et al. [38] survey the available literature on teaching introductory programming, including the choice of programming language, and conclude that there is too little evidence to support a particular approach. More recent literature surveys [24], [35] identify significant shortcomings in prior work, including an overemphasis on qualitative observations and an inability to replicate results. These findings reinforce the need for a large-scale, systematic study such as ours.

a) Choice of Programming Language: Zelle [52] argues that scripting languages, specifically Python, are better than systems languages for teaching introductory computer science courses. Gupta [19] proposes requirements for introductory programming languages and elaborates on the shortcomings of contemporary languages that make them less suitable for beginners. Mannila and de Raadt [34] offer seventeen criteria to guide the decision of which programming language to use in introductory programming courses and conclude that languages with simple syntax and semantics (specifically Eiffel and Python) are best. Stefik and Siebert [42] support these assessments: languages with C-style syntax (specifically Java and Perl) are comparable to novices in difficulty as a randomly-generated programming language whereas students perform better with languages that deviate (e.g., Python and Ruby).

b) Python in Introductory Programming Courses: A number of studies have examined using Python in an introductory programming course (i.e., CS1). Students who learned Python instead of C++ in CS1 performed comparably in a C++-based CS2 course [18] nor was there a negative effect in subsequent courses [17]. Koulouri et al. [28] found that students' understanding of programming concepts increased when using Python instead of Java in an introductory programming course. Wainer and Xavier [44] compared teaching an introductory programming course in C and Python. They found no negative consequences as a result of the change and, more specifically, that using Python decreased the number of submissions per completed assignment, increased the proportion of completed assignments, increased exam scores, and decreased the failure rate. Conversely, Alzahrani et al. [2] report that the "struggle rate" for students learning Python was significantly higher than students learning C++. While these studies focus on different textual programming languages, our work differs due to the change from a visual programming language (i.e., RAPTOR) to a textual programming language (i.e., Python).

c) Visual Programming Languages: Within the past decade, visual programming (e.g., Alice [12] and Scratch [33]) has grown increasingly popular. These programming environments aim to minimize the barrier to entry by scaffolding novices while still providing the expressive power of traditional programming languages [45]. Perhaps more importantly, block-based programming has been shown to increase engagement and interest in programming among traditionally underrepresented groups in computing fields [27]. Weintrop and Wilensky [47]–[49] have studied how the modality influences students learning to programming. After five weeks, students learning to program overwhelmingly found a block-based paradigm to be easier than a text-based programming language [47] and demonstrated greater learning and showed more interest in future computing courses [48]. In contrast, students using

text-based programming viewed it as more similar to what professionals do (more authentic) and believed it to be more effective at improving their programming ability [48]. After fifteen weeks, though, students who transitioned from block-based to text-based programming performed comparably to those who used text-based programming exclusively [49]. The most likely explanation for this result is quicker mastery of programming concepts when using block-based programming.

While our work is similar in spirit to Weintrop and Wilensky's [47]–[49], there are several key differences. First, our students are undergraduates rather than in high school. As many as 40% of our students have previously taken a course related to computing, and at least 10% have already been exposed to programming. Our setting also does not facilitate the same exploration and creativity that appeal to traditionally underrepresented groups in computing (although, in keeping with best practices [15], [43], we try to craft assignments that are personally relevant or address societal issues). Second, we do not transition from a block to text-based modality (i.e., teach students multiple programming languages) during the semester. Based on prior results [49], we expect our students to perform comparably at the end of the semester. Finally, unlike the prior work that compares text-based programming languages in introductory programming courses, we are not aware of similar studies that examine this issue for visual programming languages or compare a visual programming language (e.g., RAPTOR) to Python.

VI. Conclusion

Changing the programming language in our introductory computing course is the most significant change to the course in nearly two decades. While students' final scores in the course were comparable regardless of learning a textual programming language (i.e., Python) or a visual programming language (i.e., RAPTOR), students' overwhelming preferred to learn Python.

In this work, we consider factors that might predict students' achievement, including performance on standardized tests, in mathematics courses, and in courses that satisfy general education requirements. Our results are mixed but suggest that a visual programming language tends to "level the playing field" insofar as students' performance is less strongly correlated with the aforementioned factors (although the magnitude of the effect is small). Furthermore, students learning a textual programming language did not fare better in subsequent courses, including those focused on programming, which is consistent with prior research [49]. To summarize, students who are expected to do well tend to do well in undergraduate coursework, independent of the programming language taught in an introductory computing course—i.e., good students are generally good students.

Our findings underscore the need to articulate the objective of an introductory computing course. If equity is a concern, then visual programming appears to have no drawback apart from its perceived lack of authenticity. Conversely, students who are interested in or even ambivalent to studying computer science may be more interested in learning a textual programming language because they will be able to use it in the future. Thus, a "one size fits all" approach may promote equality rather than equity, making contextualized computing [20] better suited to both engaging and retaining individuals from a wide variety of backgrounds.

Acknowledgments

We thank all the faculty in the Department of Computer and Cyber Sciences at the United States Air Force Academy for their contributions to Comp Sci 110.

This work is partly sponsored by the Air Force Office of Scientific Research (AFOSR) under Grant FA9550-20-S-0003 as part of the Dynamic Data and Information Processing portfolio of Dr. Erik Blasch. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the United States Air Force Academy, the Air Force, the Department of Defense, or the U.S. Government.

References

[1] J. Alisaari, T. Turunen, A. Kajamies, M. Korpela, and T.-R. Hurme, "Reading comprehension in digital and printed texts," *L1-Educational Studies in Language and Literature*, vol. 18, no. 1, pp. 1–18, Dec. 2018.

[2] N. Alzahrani, F. Vahid, A. Edgcomb, K. Nguyen, and R. Lysecky, "Python Versus C++: An Analysis of Student Struggle on Small Coding Exercises in Introductory Programming Courses," in *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '18. New York, NY, USA: Association for Computing Machinery, 2018, pp. 86–91.

[3] A. Annamaa, "Thonny: A Python IDE for Learning Programming," in *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*, ser. ITiCSE '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 343.

[4] R. J. Barker and E. A. Unger, "A Predictor for Success in an Introductory Programming Class Based upon Abstract Reasoning Development," in *Proceedings of the Fourteenth SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE '83. New York, NY, USA: Association for Computing Machinery, 1983, pp. 154–158.

[5] R. A. Bartsch, "Designing SoTL Studies—Part I: Validity," *New Directions for Teaching and Learning*, vol. 2013, no. 136, pp. 17–33, 2013.

[6] ——, "Designing SoTL Studies—Part II: Practicality," *New Directions for Teaching and Learning*, vol. 2013, no. 136, pp. 35–48, 2013.

[7] B. A. Becker and T. Fitzpatrick, "What Do CS1 Syllabi Reveal About Our Expectations of Introductory Programming Students?" in *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1011–1017.

[8] S. Bergin and R. Reilly, "Predicting Introductory Programming Performance: A multi-institutional multivariate study," *Computer Science Education*, vol. 16, no. 4, pp. 303–323, 2006.

[9] M. C. Carlisle, T. A. Wilson, J. W. Humphries, and S. M. Hadfield, "RAPTOR: Introducing Programming to Non-Majors with Flowcharts," *Journal of Computing Sciences in Colleges*, vol. 19, no. 4, pp. 52–60, Apr. 2004.

[10] ——, "RAPTOR: A Visual Programming Environment for Teaching Algorithmic Problem Solving," in *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE '05. New York, NY, USA: Association for Computing Machinery, 2005, pp. 176–180.

[11] J. P. Cohoon and L. A. Tychonievich, "Analysis of a CS1 Approach for Attracting Diverse and Inexperienced Students to Computing Majors," in *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '11. New York, NY, USA: Association for Computing Machinery, 2011, pp. 165—170.

[12] S. Cooper, W. Dann, and R. Pausch, "Alice: A 3-D Tool for Introductory Programming Concepts," *Journal of Computing Sciences in Colleges*, vol. 15, no. 5, pp. 107—116, Apr. 2000.

[13] J. M. Dewar, *Gathering and Analyzing Evidence*, 1st ed. Mathematical Association of America, 2015, vol. 83, pp. 19–44.

[14] B. DiSalvo, "Graphical Qualities of Educational Technology: Using Drag-and-Drop and Text-Based Programs for Introductory Computer Science," *IEEE Computer Graphics and Applications*, vol. 34, no. 6, pp. 12–15, 2014.

[15] D. C. Edelson and D. M. Joseph, "The Interest-Driven Learning Design Framework: Motivating Learning through Usefulness," in *Proceedings of the 6th International Conference on Learning Sciences*, ser. ICLS '04. International Society of the Learning Sciences, 2004, pp. 166—173.

[16] A. Edgcomb, F. Vahid, R. Lysecky, and S. Lysecky, "Getting Students to Earnestly Do Reading, Studying, and Homework in an Introductory Programming Class," in *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE '17. New York, NY, USA: Association for Computing Machinery, 2017, pp. 171––176.

[17] R. J. Enbody and W. F. Punch, "Performance of Python CS1 Students in Mid-Level Non-Python CS Courses," in *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 520–523.

[18] R. J. Enbody, W. F. Punch, and M. McCullen, "Python CS1 as Preparation for C++ CS2," in *Proceedings of the 40th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 116–120.

[19] D. Gupta, "What is a Good First Programming Language?" *XRDS*, vol. 10, no. 4, p. 7, Aug. 2004.

[20] M. Guzdial, "Teaching Computing to Everyone," *Communications of the ACM*, vol. 52, no. 5, p. 31–33, May 2009.

[21] ——, *Why Is It So Hard to Learn to Program?* Sebastopol, CA: O'Reilly Media, Inc., 2010, pp. 111–124.

[22] ——. (2011, Jan.) Predictions on Future CS1 Languages. Computing Education Research Blog (blog). [Online]. Available: https://computinged.wordpress.com/2011/01/24/predictions-on-future-cs1-languages/

[23] J. J. Heckman, "Sample Selection Bias as a Specification Error," *Econometrica*, vol. 47, no. 1, pp. 153–161, 1979.

[24] S. Heckman, J. C. Carver, M. Sherriff, and A. Al-Zubidy, "A systematic literature review of empiricism and norms of reporting in computing education research literature," *ACM Transactions on Computing Education (TOCE)*, vol. 22, no. 1, pp. 1–46, 2021.

[25] Joint Task Force on Computing Curricula, Association for Computing Machinery (ACM), 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.

[26] R. P. Jones, D. Cooper, D. Friedman, R. Holt, and P. Robinson, "Issues in the Choice of Programming Language for CS 1 (Abstract)," in *Proceedings of the Twenty-Fourth SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE '93. New York, NY, USA: Association for Computing Machinery, 1993, p. 301.

[27] C. Kelleher, R. Pausch, and S. Kiesler, *Storytelling Alice Motivates Middle School Girls to Learn Computer Programming*. New York, NY, USA: Association for Computing Machinery, 2007, pp. 1455—1464.

[28] T. Koulouri, S. Lauria, and R. D. Macredie, "Teaching Introductory Programming: A Quantitative Evaluation of Different Approaches," *ACM Transactions on Computing Education*, vol. 14, no. 4, Dec. 2015.

[29] L. Lambert, "Factors That Predict Success in CS1," *Journal of Computing Sciences in Colleges*, vol. 31, no. 2, pp. 165–171, Dec. 2015.

[30] N. Latini, I. Bråten, and L. Salmerón, "Does reading medium affect processing and integration of textual and pictorial information? A multimedia eye-tracking study," *Contemporary Educational Psychology*, vol. 62, 2020.

[31] R. R. Leeper and J. L. Silver, "Predicting Success in a First Programming Course," in *Proceedings of the Thirteenth SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE '82. New York, NY, USA: Association for Computing Machinery, 1982, pp. 147–150.

[32] A. Luxton-Reilly, "Learning to Program is Easy," in *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*, ser. ITiCSE '16. New York, NY, USA: Association for Computing Machinery, 2016, pp. 284–289.

[33] J. Maloney, M. Resnick, N. Rusk, B. Silverman, and E. Eastmond, "The Scratch Programming Language and Environment," *ACM Transactions on Computing Education*, vol. 10, no. 4, Nov. 2010.

[34] L. Mannila and M. de Raadt, "An Objective Comparison of Languages for Teaching Introductory Programming," in *Proceedings of the 6th Baltic Sea Conference on Computing Education Research: Koli Calling 2006*, ser. Baltic Sea '06. New York, NY, USA: Association for Computing Machinery, 2006, pp. 32–37.

[35] R. P. Medeiros, G. L. Ramalho, and T. P. Falcão, "A systematic literature review on teaching and learning introductory programming in higher education," *IEEE Transactions on Education*, vol. 62, no. 2, pp. 77–90, 2018.

[36] F. Mosteller and R. Boruch, *Evidence Matters: Randomized Trials in Education Research*. Brookings Institution Press, 2002.

[37] D. L. Parnas, "On the Criteria to Be Used in Decomposing Systems into Modules," *Communications of the ACM*, vol. 15, no. 12, pp. 1053–1058, Dec. 1972.

[38] A. Pears, S. Seidman, L. Malmi, L. Mannila, E. Adams, J. Bennedsen, M. Devlin, and J. Paterson, "A Survey of Literature on the Teaching of Introductory Programming," in *Working Group Reports on ITiCSE on Innovation and Technology in Computer Science Education*, ser. ITiCSE-WGR '07. New York, NY, USA: Association for Computing Machinery, 2007, pp. 204—223.

[39] K. Quille and S. Bergin, "Programming: Predicting Student Success Early in CS1. A Re-validation and Replication Study," in *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, ser. ITiCSE 2018. New York, NY, USA: Association for Computing Machinery, 2018, pp. 15–20.

[40] N. Rountree, J. Rountree, A. Robins, and R. Hannah, "Interacting Factors That Predict Success and Failure in a CS1 Course," in *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education*, ser. ITiCSE-WGR '04. New York, NY, USA: Association for Computing Machinery, 2004, pp. 101–104.

[41] R. H. Sloan and P. Troy, "CS 0.5: A Better Approach to Introductory Computer Science for Majors," in *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE '08. New York, NY, USA: Association for Computing Machinery, 2008, pp. 271–275.

[42] A. Stefik and S. Siebert, "An Empirical Investigation into Programming Language Syntax," *ACM Transactions on Computing Education*, vol. 13, no. 4, Nov. 2013.

[43] K. Treu and A. Skinner, "Ten Suggestions for a Gender-Equitable CS Classroom," *SIGCSE Bulletin*, vol. 34, no. 2, pp. 165—167, Jun. 2002.

[44] J. Wainer and E. C. Xavier, "A Controlled Experiment on Python vs C for an Introductory Programming Course: Students' Outcomes," *ACM Transactions on Computing Education*, vol. 18, no. 3, Aug. 2018.

[45] D. Weintrop, "Block-Based Programming in Computer Science Education," *Communications of the ACM*, vol. 62, no. 8, pp. 22–25, Jul. 2019.

[46] D. Weintrop, H. Killen, and B. Franke, "Blocks or Text? How programming language modality makes a difference in assessing underrepresented populations," in *Rethinking Learning in the Digital Age: Making the Learning Sciences Count, 13th International Conference of the Learning Sciences (ICLS) 2018*. London, UK: International Society of the Learning Sciences, Inc., 2018, pp. 328–335.

[47] D. Weintrop and U. Wilensky, "To Block or Not to Block, That is the Question: Students' Perceptions of Blocks-Based Programming," in *Proceedings of the 14th International Conference on Interaction Design and Children*, ser. IDC '15. New York, NY, USA: Association for Computing Machinery, 2015, pp. 199–208.

[48] ——, "Comparing Block-Based and Text-Based Programming in High School Computer Science Classrooms," *ACM Transactions on Computing Education*, vol. 18, no. 1, Oct. 2017.

[49] ——, "Transitioning from introductory block-based and text-based environments to professional programming languages in high school computer science classrooms," *Computers & Education*, vol. 142, Dec. 2019.

[50] B. L. Welch, "The Generalization of 'Student's' Problem When Several Different Population Variances Are Involved," *Biometrika*, vol. 34, no. 1-2, pp. 28–35, 01 1947.

[51] W. Wulf and M. Shaw, "Global Variable Considered Harmful," *SIGPLAN Notices*, vol. 8, no. 2, pp. 28—34, Feb. 1973.

[52] J. M. Zelle, "Python as a First Language," in *Proceedings of 13th Annual Midwest Computer Conference*, vol. 2, 1999.