



BlockPy: An Open Access Data-Science Environment for Introductory Programmers

Austin Cory Bart, Javier Tibau, Eli Tilevich, Clifford A. Shaffer, and Dennis Kafura, Virginia Tech

Non-computer science majors often struggle to find relevance in traditional computing curricula that tend to emphasize abstract concepts, focus on nonpractical entertainment, or rely on decontextualized settings. BlockPy, a web-based, open access Python programming environment, supports introductory programmers in a data-science context through a dual block/text programming view.

As computing becomes pervasive across all fields, professionals increasingly need to learn computing skills in addition to their core domain knowledge. General-education computing curricula at the collegiate level (for example, courses in computational thinking) are scaling, massive open online courses are flourishing, and many professionals and students are pursuing informal learning experiences on their own. However, many learners often have little experience with computing and are uncertain as to how computing benefits their long-term career goals. Not only

do learners need special scaffolding unique to their abilities and motivational levels, but they also need fewer barriers to accessing these materials. We propose a new tool to better serve this population: BlockPy, an open access, web-based Python environment for data science that supports introductory learners with guided instruction and an accessible interface (www.blockpy.com).

Modern approaches to contextualizing introductory courses have focused on making the experience fun and interesting, with an emphasis on game design and media computation. However, student motivation is a complex



Multimedia: BlockPy in Action

Watch a video demonstration of BlockPy's main features by visiting *Computer's* multimedia page (www.computer.org/computer-multimedia) or the IEEE Computer Society YouTube channel (www.youtube.com/user/ieeeComputerSociety).

construct dependent on more than just situational interest. Holistic models of motivation suggest that students also need to feel that the material is useful to learn and that long-term career goals are satisfied.¹ Contexts like media computation are not always perceived as authentically useful for non-computer science majors.²

We believe data science is a motivating context that can appeal to students in a different way, thanks to the widespread need for data processing in other majors. In a previous work, we reported on the affordances and impacts of data science as a learning context.³ Students often study computing to learn how to manage the dizzying quantities of data being stored and analyzed in a discipline or for a specific self-derived project. By grounding the content in this context, students can be more easily convinced of computing's relevance and more clearly understand how the materials fit together. By aligning the context with students' long-term needs, students learn skills that are more relevant to those needs. Finally, data science as a context naturally lends itself to teaching topics related to structured data, iteration, and other core material, making it pedagogically valuable to computer science instructors.

PYTHON AND BLOCKS

Python has become one of the most popular introductory programming languages⁴ because of its simple syntax and impressive power. It includes strong support for data science, as seen with popular libraries like Matplotlib. Python requires little code to accomplish interesting things, so novices are not bogged down with complex syntactical details. Its widespread use in introductory classes

and industry further motivated our choice to focus on Python.

Any kind of programming is a challenge to beginners, as coding is known as the “most powerful, but least usable human-computer interface ever invented.”⁵ Block-based languages (such as Scratch and AppInventor) have been shown to mitigate the start-up time for students to begin programming and accomplish tasks.^{6,7} By providing structure and an immediate view into the entire user interface of a language, blocks greatly benefit introductory learners.

There are several environments available today that let students and instructors write Python in the browser, including CodeSkulptor,⁸ Pythy,⁹ and Online Python Tutor (OPT).¹⁰ BlockPy stands on the shoulders of giants, integrating features inspired by these environments and introducing novel ones. But none of these existing Python environments helps students with the transition to textual programming languages.

BlockPy was designed to provide dual support for block-based and text-based code authoring. At any time, students can switch freely between a block-based view of their code and a traditional text-based view. This powerful feature is inspired by Pencil Code, which uses its own Logo language.¹¹ Similar implementations have been successful as a fading scaffold for students.¹²

BlockPy extends Pythy's⁹ support for “assignments”—problems that integrate presentation with assessment. However, Pythy only supports traditional unit testing to provide students with feedback, while BlockPy provides an API for code analysis and free-form text guidance that instructors can configure to give helpful suggestions to their students. Furthermore, Pythy has limited support for data science, whereas BlockPy has

a rich library of data sources and a Matplotlib-based plotting API.

CodeSkulptor, Pythy, and BlockPy all use the same internal engine for running Python code (Skulpt). Although CodeSkulptor has an extensive API for creating user interfaces and games, it provides a nonstandard library that does not aid in the transition to serious programming environments. BlockPy's philosophy is to maintain approximate compatibility with real systems. Instead of a custom plotting API, for instance, BlockPy mimics the Matplotlib interface.

OPT has proven to be a useful tool for visualizing program state. However, it provides a depth of detail that can overwhelm introductory students (for example, it uses terminology such as frames and objects, which might be foreign to beginners). BlockPy's state explorer does not attempt to match OPT's thoroughness, but instead provides a helpful yet simple picture of program state. Additionally, BlockPy avoids OPT's server dependency by relying on Skulpt, which runs in the browser.

BLOCKPY OVERVIEW

The primary design goals of BlockPy are to

- ▶ reduce barriers to learning programming;
- ▶ promote authenticity by empowering students to solve real-world problems;
- ▶ promote maturity by faded scaffolds (for example, transitioning from blocks to text); and
- ▶ minimize the need for help from human instructors.

Open source, open access

BlockPy aims to be a highly accessible, web-based platform for anyone to learn

ADVANCES IN LEARNING TECHNOLOGIES

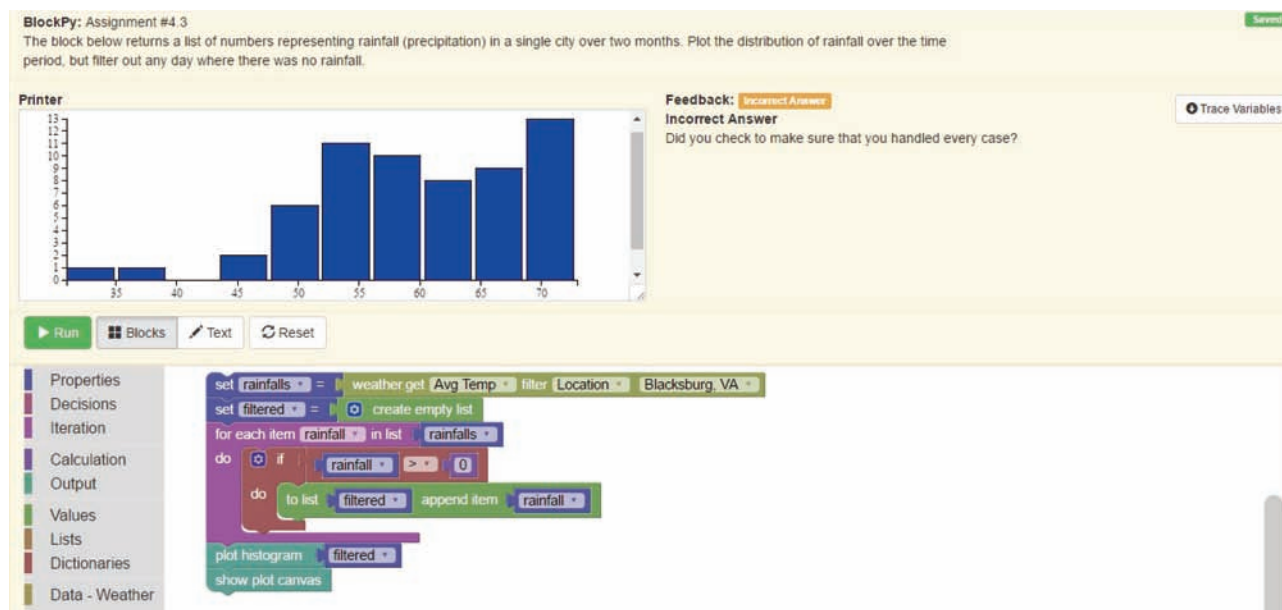


FIGURE 1. A screenshot of the BlockPy environment in action as a student completes a programming assignment. Note the current student solution in progress at the bottom and the most recent feedback shown in the upper-right box.

how to program. All code is open source and leverages a number of open source libraries. No registration is needed to use the software, although users can gain benefits from free registration, such as managing classrooms. BlockPy also provides guided learning materials to be shared by educators.

The BlockPy editor (see Figure 1) continuously stores user code as it is entered. Logs are stored at the key-stroke level for future program analysis, which we will describe later. The latest version of the user's code is therefore available between sessions. When operating in offline mode, the code is stored in the LocalStorage browser object; when the connection is reestablished, synchronization is performed.

Python execution

The BlockPy system is built to work offline, making it ideal for places where Internet connectivity is unreliable. Code execution is achieved through a modified instance of the Skulpt JavaScript library. Skulpt is a full Python parser and compiler, supporting almost all Python language features by generating JavaScript code—including partial support for the rich Python standard library. The Skulpt execution environment resides

entirely within users' browsers, so there is no reliance on an external server beyond the initial page load.

Block-based Python

To support introductory learners as they grapple with Python syntax, the initial BlockPy interface is block-based and uses the popular Blockly JavaScript library. Language features (iteration, decision, variable assignment and access, and so on) are contained in a toolbox on the left side of the interface, from which users drag and drop blocks onto a canvas. BlockPy's block interface only generates syntactically valid Python code, enforced by the "snapping" connectors of the blocks (although it is possible to generate semantically incorrect code, which will be discussed later). This block interface is synchronized with a text interface.

An important question for a block-based representation of Python is how many language details should be exposed and at what rate. A rarely used feature of for loops in Python is to contain an else clause that is executed upon successful completion of the loop (that is, when it is not prematurely escaped using a break statement). This advanced language feature is similar to a finally statement

with exceptions. However, if an else clause were made available to those just starting out with iteration, they would likely confuse the concept with the conditional else clause used in if statements. Cognitive load is harsh for beginners, and the user interface should avoid exposing unnecessary details where possible.

While hiding else bodies in a for loop is a clear case, there are subtler examples. It can be difficult to recognize when a learner is ready to use parallel assignment and therefore should be able to specify multiple variables on the left side of an assignment block. A block-based language forces teachers to make important decisions about how to expose language features. For future BlockPy work, we are experimenting with exposing language features at different rates, which can be adjusted by the instructor, so the system can expose students to a progressively more accurate language model.

Adaptive guided practice


One of BlockPy's most powerful features is its interactive guided feedback. A limitation of programming environments like Snap! is that they are not pedagogically interactive—students

completing an assignment in the system are not guided to success. The learner must decide when they have completed their program, and whether it meets the specification. For independent informal learners, this requires high levels of self-regulation and metacognition. BlockPy's adaptive elements follow an Expert model; when students run their code, it is checked against instructor-provided logic. If students' code fails for some reason, they are offered a suggestion. If the code is correct, the system presents a green "Complete" mark, which has been found to have motivating power.

In BlockPy's instructor mode, teachers provide a problem instance. First, a WYSIWYG (what you see is what you get) rich-text editor edits the problem description, supporting any valid HTML content (such as images and links). Second, the instructor provides code in special canvases that affect students' experience, using the same text/block interface that students use. First is the starting code, which is shown to students when they begin the problem (to avoid a blank canvas). The second instructor code defines interactive feedback, which can access students' code, their final output, and a complete trace of their program's state. The checking system can declare the code to be correct or display an HTML string that is rendered as user feedback. The instructor is free to write whatever logic they want, such as searching for a specific abstract syntax tree (AST) element, testing the outputs on the console, or walking through the program state to satisfy invariants. An API for common checks is evolving based on common use cases, such as parsing the program's AST to ensure that they are not calling forbidden built-in Python functions.

We are also exploring interventions for instructors beyond rendering textual feedback, such as displaying a pop-up dialog with an embedded instructional video or alerting an instructor to provide just-in-time instruction to a particular struggling student. In addition, we are experimenting with a new system for modeling students' misconceptions related to programming in an effort to

who frequently move the same blocks without progressing in the problem objectives might indicate taking longer on the problem than other users. Alternatively, students who pick a decision block to complete a problem about iteration might need extra attention. By proving or disproving such hypotheses, we can improve the system's automatic feedback and provide more individualized support.



BLOCKPY IS A HIGHLY ACCESSIBLE, WEB-BASED PLATFORM FOR ANYONE TO LEARN HOW TO PROGRAM.

provide better guidance that is tuned to students' specific mistakes.

Program analysis for deeper learning

BlockPy uses simple program analysis techniques to find both general mistakes that novices tend to make and problem-specific errors. Beginners often fail to understand the true purpose of certain variables and incorrectly include them in their code by mimicking an example, but BlockPy can identify these variables and raise an error by performing simple variable liveness analyses. Most modern editors feature this kind of analysis, but usually trust that the user has a reason for adding the variable and thus respond passively.

Because BlockPy securely records the history of users' programs and logs interface interactions, it can mine this repository of code to infer common patterns that suggest undesirable learner behavior. For example, users

Learning Technology Interoperability

BlockPy supports the Learning Technology Interoperability (LTI) protocol—a mechanism by which instructors can embed questions in their existing course management software (such as Canvas and OpenEdX) and receive assignment outcomes (such as a grade). A typical learner uses BlockPy without ever being aware of LTI. Instructors using the system obtains a secret key and configuration URL that is used within their learning management system (LMS). Students on a course website might use BlockPy without registering for a BlockPy account: the first time they log in through their LMS's provided link, they are invisibly registered in the system with a regular account through additional information from the LMS. As students complete work, assignment progress is reported back to the LMS. Instructors use a special

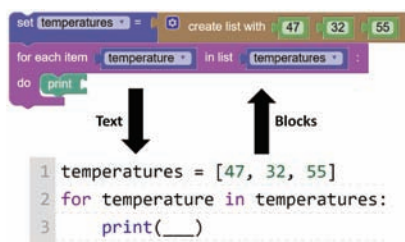


FIGURE 2. User perspective of block/text transition. The top snippet of block code demonstrates creating a list and iterating through it with a loop. The bottom snippet is the generated Python textual code. The arrows connecting the two indicate the bidirectional nature of the code.

interactive menu for managing exercises associated with a course.

Data science blocks

BlockPy focuses on data science as its primary context, so it includes blocks for working with data. BlockPy supports a subset of the popular Matplotlib library and extends it with connections to the CORGIS (Collection of Real-time, Giant, Interesting, Situated datasets) project.¹³ The Matplotlib API, for example, provides a `plot()` function to create simple line plots. By mimicking Matplotlib, students can seamlessly shift to a serious Python programming environment without loss of code.

The CORGIS project makes motivating datasets available to introductory students through simple programming libraries.¹³ These datasets are drawn from many disciplines, resulting in material meant to be universally interesting and relevant. Currently, BlockPy supports a number of different CORGIS libraries including weather data, earthquake data, US crime statistics, and a classic book dataset.

Mutual language translation

A technical contribution of this project is the mutual language translation between Blockly and Python. Blockly outputs valid Python source code, which can be passed into Skulpt to extract a JSON representation of the AST. This AST is parsed using our own Py2Block library to generate an XML representation that Blockly can render in the block view. Figure 2 demonstrates the users' experience. When a student tries to convert code with disconnected blocks, the generated Python code will be filled in with triple underscores. These underscores (usually a valid variable name in Python) will trigger a runtime error.

Blockly already supports compilation of its blocks to Python, JavaScript, PHP, and Dart. However, this multiple-language support leads to reduced isomorphism—each language has different syntax for their common operations, and it is impossible to create a fully featured block language with a one-to-one mapping to them all. For example, JavaScript has no support for parallel assignment—a commonly used feature in Python—whereas Python does not have a unary increment operator. Blockly's syntax and vocabulary are descended from Logo.

Instead of trying to satisfy multiple languages, we dropped support for other languages in favor of a more fully featured mapping to Python. This requires minor changes that introduce Python-centric syntax details: function blocks are labeled `define`, assignment blocks have an `=` symbol, and the `add item to list` block is renamed to `append`. Blockly has also been extended with new language features, including dictionary access and creation.

Eventually, the BlockPy interface should offer a complete isomorphic

mapping to Python. However, a number of complications need to be resolved first. For instance, Python uses square brackets for both list indexing and dictionary access. There is a strong desire to differentiate between these types of access, visible in the block view as two distinct kinds of blocks (`get ith element of list` versus `get key from dict` blocks). However, it is computationally difficult to statically identify the usage of a given pair of brackets; sophisticated program analysis techniques are needed.

Parson's problems

Parson's problems are a special type of coding exercise where all the necessary code blocks are present, but disconnected and shuffled. These kinds of problems scaffold beginners by providing everything they need to complete the problem, reducing many of the barriers to getting started. BlockPy supports these types of problems with a special Parson's mode, where top-level blocks are shuffled in the block mode.

State explorer

BlockPy provides a state explorer, used to trace programs' execution over time. The state explorer displays information about variables and allows users to step through the code's execution, affecting what is currently printed/plotted, imported modules, and the values and types of variables.

MODEL USE CASES

Here, we consider some scenarios that describe our vision of typical BlockPy use cases. BlockPy aims to be useful in both formal and informal situations.

Independent learner

A learner independently logs into the BlockPy system and selects an introductory problem on calculating

averages using iteration: “Is the weather in Seattle above 60 degrees Fahrenheit? Print Yes or No.” The novice user is unsure what to do after reading the problem description. If the user decides to cheat by checking the current weather in Seattle and printing the literal value, the system intelligently notices that the user is missing a relevant weather block and explains that he or she needs to combine programmatic decision logic with the appropriate data source. The user then accesses the “Weather” block category and grabs the `weather.get` block, but is unsure of what to do next. When the user runs the program, the system notices that there are not any `if` statements, and suggests reading a linked chapter in an online textbook. If the user continues to struggle with integrating pieces, the system can provide increasingly detailed hints until he or she succeeds.

Classroom lesson

An instructor is using Canvas, an LTI-capable LMS, to create a series of assignments for the day. As students complete the assignments, their grades are reported to Canvas, so the instructor can monitor students’ progress and help those who are struggling to complete assignments. This information allows instructors to target underperformers with earlier interventions. The more automatic feedback that instructors make available through BlockPy, the less they need to focus on simple problems (“You were checking the temperature for the wrong city”) so they can turn their attention to students who are truly struggling (“What is iteration?”).

One-on-one tutoring

A student is struggling to write the necessary syntax for indexing

a nested dictionary (for example, a crime report broken into multiple levels, with the burglary rate for a city nested under a violent crime categorization). The student does not have a clear image of how the layered structure of data can translate into a chain of dictionary accesses. Sitting with the

of sophomores and juniors (26 percent), and fewer freshmen (14 percent).

The course content focused on teaching abstraction and algorithms. Although programming was not a primary learning objective, it was an important topic in the course for discussing higher-level objectives. During

**BLOCKPY FOCUSES ON DATA SCIENCE
AS ITS PRIMARY CONTEXT,
SO IT INCLUDES BLOCKS
FOR WORKING WITH DATA.**

student, an instructor or tutor builds up an expression accessing the data by connecting dictionary access blocks to the data block, showing the generated Python code at each step. The student can visually see how the chunks of code correlate to blocks, which helps them understand that code is not a series of symbols but a structured representation of an algorithm.

PILOT STUDY

BlockPy was piloted in a collegiate-level introductory computational thinking course with 35 students in spring 2015. These students came from a diverse range of majors, including liberal arts (57 percent), architecture (17 percent), and sciences (15 percent). There were 20 female students (57 percent) and 15 male students. The majority of students reported no prior experience in programming, and less than 17 percent reported taking a computer science advanced placement course in high school. Students were evenly distributed across years, with slightly more seniors (29 percent), equal percentages

the first third of the course, students worked with NetLogo (although they did not program in it, they used it to read code) and participated in explanatory kinesthetic activities. The students were then introduced to Python using BlockPy, where they spent roughly six classes on completing guided practice problems. The next two classes were devoted to using a regular Python environment (Spyder) to complete small programming assignments (similar to the ones completed with BlockPy). Finally, students were given eight classes to work on an individual final project in Spyder.

Methodology

Student responses to BlockPy were collected through two surveys, one given after the BlockPy section and the other given at the end of the course. The survey was composed of four-point Likert questions and open-ended qualitative questions. Figure 3 shows a selection of particularly interesting results. All conclusions from this study should be considered

ADVANCES IN LEARNING TECHNOLOGIES

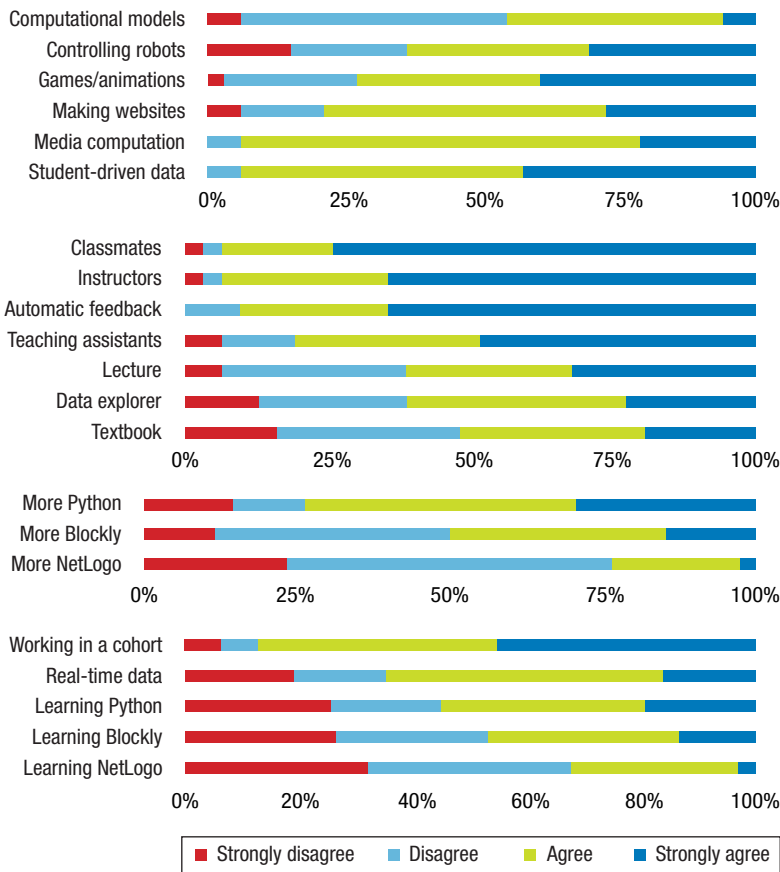


FIGURE 3. A selection of student responses from a survey on BlockPy.

preliminary because it used the first version of BlockPy.

Perceptions of BlockPy

The first survey question asked whether students wanted more time with each of the programming environments they used in the course: NetLogo, BlockPy, and Spyder. Note that BlockPy was referred to as Blockly and the Spyder environment was referred to as Python. The results suggest that students valued their experiences with BlockPy more than NetLogo, but mostly felt they were not getting enough Python experience. This is backed up by the qualitative data, where some students said “more Blockly, less Python” and others asked for “more Blockly and more Python.”

BlockPy usage

Over the six days spent using BlockPy, students were tasked with 40 classwork questions and 19 homework

questions. Students ran their code an average of four times per problem (standard deviation 1.8).

Students were asked if they felt successful when transitioning from BlockPy to Spyder. Sixty-five percent of the class agreed or strongly agreed, suggesting that a sizeable population felt uncomfortable during the transition. The original design of the mutual language translation featured the block and text views simultaneously, side by side. However, log analysis revealed that most students did not take advantage of the feature. Only five students (roughly 15 percent) used the conversion functionality at all, and fewer used it consistently. It is possible that students were observing the code as it changed, but they were not writing textual code. It is difficult to say why students did not take advantage of the feature. Our current hypothesis is that students were confused by the interface, which required manual

conversion to go from text to blocks. In our new version of BlockPy, the conversion happens automatically—simply by switching tabs—and intentional opportunities are presented for switching. Preliminary data suggests that this new interface greatly improves students’ transition.

Students were also surveyed about what most helped their learning. Peers and instructors were about on par with the automatic feedback given in BlockPy, suggesting the strong value of the system. Despite the popular response to the state explorer, relatively few students took advantage of it (11 students; roughly 31 percent). Because more than 50 percent of the class reported finding value in the data explorer, it is possible that the students benefited from instructor presentations of the tool, even if they did not take advantage of it themselves.

Data-science context

Finally, students were surveyed about their perceptions of the value of different course experiences with regard to their long-term career goals and their interest in potential contexts for introductory computing courses. Each of these contexts were briefly described—for example, the media computation context was listed as “working with pictures, sounds, and movies.”

The results suggest that students find data science to be compelling—this should be taken with a grain of salt, however, because most students have negligible experience with alternative contexts. However, our preliminary results suggest that this is an approach worth exploring further.

FUTURE WORK

BlockPy is an evolving project. We have a number of features planned to expand

support for Python and for the guided feedback API for instructors, such as leveraging more static/dynamic inference techniques to improve block rendering and error reporting.

One of the biggest values of a block-based environment is that it can immediately expose the breadth of a rich API, which greatly reduces students' dependency on documentation. However, this can also be a downside, as students might be overwhelmed by the interface's features. Thus, an open research question is deciding at what rate to expose language features.

One of the major advantages of game and animation design as an introductory context is that they make abstract concepts concrete. Further analysis is needed to determine the tradeoffs of using different contexts. BlockPy can help determine these tradeoffs by supporting alternative contexts, such as turtle graphics and media computation libraries.

It is difficult to derive conclusive results from our pilot study due to the small population size and the evolving nature of BlockPy. Preliminary results from in-progress studies suggest that recent improvements have overcome a number of limitations to the environment and that user feedback has dramatically improved. We are conducting follow-up studies on the logged students' code, even as we collect more data on the newest iteration. We are hopeful that BlockPy will increase its user base, providing a larger sample of learners and more meaningful data.

BlockPy is being developed in an on-demand fashion, driven by immediate course needs, but is still limited. For example, the block interface does not support a number of advanced Python features, such as an interface for writing object-oriented classes.

This does not mean that students cannot write programs featuring classes or other advanced features, as Python code using these features render in BlockPy as embedded text blocks and execute through Skulpt normally. There is no technical impediment to supporting these features—the

process is limited only by time and community interest.

We believe BlockPy represents a new paradigm for introductory learners, blending interactive support with a

ABOUT THE AUTHORS


AUSTIN CORY BART is a visiting assistant professor at Virginia Tech. His research interests include computer science education, software engineering, and program analysis. Bart received a PhD in computer science and learning sciences from Virginia Tech. Contact him at acbart@vt.edu.

JAVIER TIBAU is a PhD candidate in the Computer Science Department at Virginia Tech, where the research described in this article was performed. He is also an assistant professor of computer science at ESPOL. His research interests include human–computer interaction and computer science education. Tibau received an MSc in computer science from Universitat Politècnica de Catalunya. Contact him at jtibau@vt.edu.

ELI TILEVICH is an associate professor of computer science at Virginia Tech. His research interests include software engineering for distributed and mobile computing and computer science education. Tilevich received a PhD in computer science from Georgia Tech. He is a Senior Member of IEEE. Contact him at tilevich.vt.edu.

CLIFFORD A. SHAFFER is a professor of computer science at Virginia Tech. His research interests include computational biology—specifically, user interfaces for specifying models and computations—algorithm visualization, and computer science education. He received a PhD in computer science from the University of Maryland. Contact him at shaffer@vt.edu.

DENNIS KAFURA is a professor of computer science at Virginia Tech. He is the principal investigator on two NSF IUSE awards, both of which involved the development of a general education course in computational thinking at the university level using the BlockPy environment. Kafura's research interests include computer science education, operating systems, and software engineering. He received a PhD in computer science from Purdue University. Contact him at kafura@vt.edu.

clear path to programming maturity. By teaching in the context of data science, BlockPy provides authenticity even as it helps transition students to a more serious programming environment. Research using BlockPy will help answer crucial questions about the value of data science and blocks. Our hope is that BlockPy's open nature can encourage learners from diverse fields to engage with computing in a way that will lead to a computing-rich future for a larger population. 

ACKNOWLEDGMENTS

We gratefully acknowledge the support of the National Science Foundation under grants DGE-0822220, DUE-1444094, and DUE-1624320.

REFERENCES

1. B.D. Jones, "Motivating Students to Engage in Learning: The MUSIC Model of Academic Motivation," *Int'l J. Teaching and Learning in Higher Education*, vol. 21, no. 2, 2009, pp. 272–285.
2. M. Guzdial and A.E. Tew, "Imagineering Inauthentic Legitimate Peripheral Participation: An Instructional Design Approach for Motivating Computing Education," *Proc. 2nd Int'l Workshop Computing Education Research (ICER 06)*, 2006, pp. 51–58.
3. A.C. Bart et al., "Computing with CORGIS: Diverse, Real-World Datasets for Introductory Computing," *Proc. 48th ACM Technical Symp. Computer Science Education (SIGCSE 17)*, 2017, pp. 57–62.
4. P. Guo, "Python Is Now the Most Popular Introductory Teaching Language at Top US Universities," blog, *Comm. ACM*, 7 Jul. 2014; cacm.acm.org/blogs/blog-cacm/176450-python-is-now-the-most-popular-introductory-teaching-language-at-top-u-s-universities/fulltext.
5. A. Ko, "Programming Languages Are the Least Usable, but Most Powerful Human-Computer Interfaces Ever Invented," blog, 25 Mar. 2014; blogs.uw.edu/ajko/2014/03/25/programming-languages.
6. T.W. Price and T. Barnes, "Comparing Textual and Block Interfaces in a Novice Programming Environment," *Proc. 11th Ann. Int'l Conf. Int'l Computing Education Research (ICER 15)*, 2015, pp. 91–99.
7. D. Weintrop and U. Wilensky, "Using Commutative Assessments to Compare Conceptual Understanding in Blocks-Based and Text-Based Programs," *Proc. 11th Ann. Int'l Conf. on Int'l Computing Education Research (ICER 15)*, 2015, pp. 101–110.
8. T. Tang, S. Rixner, and J. Warren, "An Environment for Learning Interactive Programming," *Proc. 45th ACM Technical Symp. Computer Science Education (SIGCSE 14)*, 2014, pp. 671–676.
9. S.H. Edwards, D.S. Tilden, and A. Allevato, "Pythy: Improving the Introductory Python Programming Experience," *Proc. 45th ACM Technical Symp. Computer Science Education (SIGCSE 14)*, 2014, pp. 641–646.
10. P. Guo, "Online Python Tutor: Embeddable Web-Based Program Visualization for CS Education," *Proc. 44th ACM Technical Symp. Computer Science Education (SIGCSE 13)*, 2013, pp. 579–584.
11. D. Bau, M. Dawson, and A. Bau, "Using Pencil Code to Bridge the Gap Between Visual and Text-Based Coding (Abstract Only)," *Proc. 46th ACM Technical Symp. Computer Science Education (SIGCSE 15)*, 2015, p. 706.
12. Y. Matsuzawa et al., "Language Migration in Non-CS Introductory Programming Through Mutual Language Translation Environment," *Proc. 46th ACM Technical Symp. Computer Science Education (SIGCSE 15)*, 2015, pp. 185–190.
13. A.C. Bart, "Situating Computational Thinking with Big Data: Pedagogy and Technology (Abstract Only)," *Proc. 46th ACM Technical Symp. Computer Science Education (SIGCSE 15)*, 2015, p. 719.



IEEE Intelligent Systems

THE #1 ARTIFICIAL INTELLIGENCE MAGAZINE!

IEEE Intelligent Systems delivers the latest peer-reviewed research on all aspects of artificial intelligence, focusing on practical, fielded applications. Contributors include leading experts in:

- Intelligent Agents • The Semantic Web
- Natural Language Processing
- Robotics • Machine Learning

Visit us on the web at www.computer.org/intelligent