Encouraging the Use of Built-in Language Features for Learning Control Flow

Student researcher:
Michael Mobilio (IT)

Faculty mentor:
Michael J. Lee, PhD (Informatics)

Abstract

This research attempts to develop a learning curriculum that encourages the use of specific control flow patterns in the JavaScript and Python programming languages. This is a technical approach that benefits the user by letting them construct, visualize, and follow specific control flow patterns through the use of the built-in features from their choice of programming language. This research aims to improve students' understanding and recognition of better control flows.

1. Background

The built-in features of a programming language are the building blocks of a computer program. These blocks are used to control the flow of a program. “Control flow” is the order that statements are evaluated in a program. The simplest methods of controlling a program’s flow are through the use of conditionals such as `if` and `switch` statements, and the use of loops such as `for` and `while` loops (Buhr, 2016). For example, MIT’s Scratch allows users to experiment with control flow through the use of drag-and-drop code blocks while writing their code to control characters on the screen. However, while Scratch is a visual language, it does not clearly visualize exactly what part of the code is running at any given time. Assisting learners, especially novices, better understand the step-by-step process of a running program may help them better understand control flow, and in the process, how different statements within a program can change the course of this flow during program execution.

Figure 1. Scratch, Visual/Block Programming Language

Players drag-and-drop code blocks into the scripts panel on the right, and then click ‘run’ to watch their code evaluate on the left screen by displaying an animated scene that the script controls. The Scratch programming language allows for the same control flow ideals as most other programming languages, for example, Scratch uses `when` statements and `forever` loops that function similarly to `if` statements and infinite loops, respectively.

Some other programming tools, such as Python Tutor (Guo, 2013) and Coral (Coral, n.d.), do a better job of helping users visualize the control flow of a program by providing visual aids like flow charts

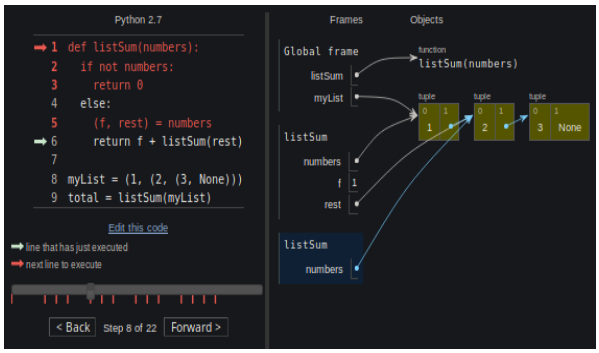
and line evaluation indicators. By allowing users to alter the flow of their program, it is possible to directly give non-threatening feedback on control flow choices. With a simple development environment inside a container similar to Python Tutor, users are encouraged to view the real-time steps their program takes, and the results that occur based on their code.

This method is in stark contrast to a typical compiler run-through, in which a user runs a program, and all errors are reported after-the-fact (known as a “compile-time error”). A compiler will typically fail to run the program and instead, return lines of ilegible jargon which a novice programmer will struggle to understand. The error could be as simple as a missing semicolon, but the user may have a hard time deciphering that from an unintelligible, multi-line error message. Although more advanced programmers may understand that these types of errors are common and can typically be fixed with minor debugging, a novice programmer, who is prone to errors since they are learning, can quickly become overwhelmed and discouraged by these errors.

With a visual control flow, the frustration of finding where a problem occurred in the code is removed as the user can visually and easily trace back the steps that caused the error or bug in the first place. Program visualization has clear benefits but is often not used into introductory programming curriculum (Karnalim & Ayub, 2017) and instead typically available in debugging tools designed for more advanced programmers. Furthermore, little is known how to create directed educational content that highlights the use of specific control flow patterns to better teach novices programming concepts.

Figure 2. Python Tutor

Users can watch their code evaluate step-by-step and see detailed information about the current state of their program. The information is useful to see the values of variables and how they change during the program’s evaluation. I want to implement a visualizer similar to Python Tutor’s and also include a flow chart generator.



2. Project Proposal

The intention of this project is to design and develop a curriculum (a collection of levels, each with a specific learning objective) that includes thought-provoking challenges to encourage players to follow and visualize specific control flow patterns. These challenges will be implemented into level designs that incentivizes short and simple code through the use of control flow. To solve a level, players will be challenged to code and debug a solution in their choice of JavaScript or Python, two of the most popular programming languages (Chan, 2019). Players will be able to debug their code to level completion with the usage of hints, code examples, and interactive debugging. An indicator will display where the program is in its progress, helping the player visualize the control flow of their program.

Novice programmers will often unknowingly choose a tedious control flow implementation, out

of not knowing or lack of understanding the programming language features. For example, in Python: using the `for` loop statement can improve the repetitive task of collecting user input, converting it to an integer type, and adding it to the sum.

Figure 3. Worse Flow; Repetitive Adding Machine

```
# Adding machine to add three numbers together
sum = 0
number1 = input('Enter a number: ')
if number1.isnumeric():  # Validate number
    sum += int(number1)  # Add to sum

number2 = input('Enter a number: ')
if number2.isnumeric():  # Validate number
    sum += int(number2)  # Add to sum

number3 = input('Enter a number: ')
if number3.isnumeric():  # Validate number
    sum += int(number3)  # Add to sum

print(sum)
```

Figure 4. Better Flow; For Loop Adding Machine

```
# Adding machine to add three numbers together
sum = 0
for i in range(3):
    number = input('Enter a number: ')
    if number.isnumeric():  # Validate number
        sum += int(number)  # Add to sum

print(sum)
```

I will develop these features and curriculum within the Gidget[1] web application for ease of development and ease of access for the user. Gidget is a free online puzzle game designed to teach novices introductory programming concepts by solving debugging puzzles. Gidget has been used by a wide range of users around the world (Lee, 2015) and extensively in programming events for middle school students (Lee, 2019) and high school students (Lee et al. 2014) promoting diversity. By using a web platform, my project can utilize stable web technologies such as JavaScript and CodeMirror for the project's code editor while also ensuring there is a wide user base. To promote continued use of the project, users will be able to continue from any computer they want, provided that they have internet access.

3. Significance

Programming is rapidly becoming an important skill to have in almost every work environment. Now, programming and the need to be an effective programmer is a valuable skill and will become increasingly important in the workplace (Grace, 2019). Children as young as kindergarteners are familiar with computer programs (Kafai & Burke, 2014) and there are numerous activities to teach middle and high school students programming (e.g., Lee et al., 2014; Lee, 2019). Therefore, creating new and more effective ways to teach programming is imperative. I believe that using control flow as a basis to teach introductory concepts will benefit novices tremendously, as learning and understanding how a program runs step-by-step early in the learning process will continue to be helpful throughout one's experience with code, and also in learning more advanced features later on.

4. Expected Outcomes and Deliverables

This project should be expected to deliver an engaging beginner programming curriculum that focuses on encouraging the use of better control flow patterns. This project should also be expected to

[1] Gidget is a free, NSF-funded educational programming game available at: *helpgidget.org*

provide statistical data of user interaction including: solve time duration, code solution length, and choice in control flow. This data may be represented in the form of visual graphs. In addition, learners are expected to solve challenges faster and with fewer lines of code from the use of better control flow patterns, than users who choose to use less appropriate control flow patterns.

5. Project Design

Week 1
Design and mockup user interface. Begin storyboarding control flow challenges.

Week 2
Begin implementing CodeMirror editor and debugger.

Week 3
Continue implementation of debugger.

Week 4
Create interface to log user programming data to a database.

Week 5
Implement control flow challenges and work on assets needed for final implementation.

Weeks 6-7
Continue control flow challenge implementations. Add tests for control flow challenges.

Week 8
Implement other visual structures such as flow charts and call stack tables.

Week 9
Use feedback from previous week to implement desirable changes. Begin preparing presentation for symposium.

Week 10
Write final documentation for the project and troubleshoot any final issues in preparation for the symposium.

Bibliography/Works Cited

1. *Buhr, P. A. (2016). Understanding control flow : Concurrent programming using [mu]C / Peter A. Buhr.*

2. *Chan, R. (2019, January 22). The 10 most popular programming languages, according to the 'Facebook for programmers'. Retrieved from https://www.businessinsider.com/the-10-most-popular-programming-languages-according-to-github-2018-10*

3. *Coral. (n.d.). Retrieved from https://corallanguage.org/*

4. *Grace, K. (2019). Will AI Replace Humanity: A Primer for the 21st Century. Thames & Hudson.*

5. *Guo, P. J. (2013). Online python tutor: embeddable web-based program visualization for cs education. In Proceeding of the 44th ACM technical symposium on Computer science education (pp. 579-584). ACM.*

6. Kafai, Y. B., & Burke, Q. (2016). Connected code: Why children need to learn programming. Cambridge, MA: The MIT Press.

7. Karnalim, O., & Ayub, M. (2017). *The Effectiveness of a Program Visualization Tool on Introductory Programming: A Case Study with PythonTutor. CommIT (Communication and Information Technology) Journal, 11(2), 67. doi:10.21512/commit.v11i2.3704*

8. *Lee, M.J. (2015). Teaching and Engaging With Debugging Puzzles. University of Washington Dissertation (UW), Seattle, WA.*

9. *Lee, M.J. (2019). Exploring Differences in Minority Students' Attitudes Towards Computing After a One-Day Coding Workshop. ACM Innovation and Technology in Computer Science Education (ITiCSE).*

10. *Lee, M.J., Bahmani, F., Kwan, I., Laferte, J., Charters, P., Horvath, A., Luor, F., Cao, J., Law, C., Beswetherick, M., Long, S., Burnett, M., and Ko, A.J. (2014). Principles of a Debugging-First Puzzle Game for Computing Education. IEEE Visual Languages and Human-Centric Computing (VL/HCC), 57-64.*