GIFTS: Introduction to Programming with Hardware for First-Year Engineers

Dr. Darby Hewitt, Abilene Christian University

Darby Hewitt is a Professor of Electrical Engineering at Abilene Christian University, where he has taught since 2013. Dr. Hewitt has a BS in Physics from Abilene Christian University and MS and PhD degrees in Electrical and Computer Engineering from the University of Illinois at Urbana-Champaign. Dr. Hewitt teaches courses in analog circuits, digital electronics, and embedded systems. When he's not teaching about electronics or writing firmware, he's either reading fiction or playing the guitar.

Dr. Robert L Brown II, Abilene Christian University

Dr. Brown is an associate professor and Director of Mechanical Engineering at Abilene Christian University in the Department of Engineering and Physics. He received B.S. and Ph.D. degrees in Aerospace Engineering from Texas A&M University. His primary focus of research is alternative grading schemes, with secondary interests in computational fluid dynamics and finite element modeling.

Joseph Sylvester Makarewicz, Abilene Christian University

GIFTS: Introduction to Programming with Hardware for First-Year Engineers

Abstract

This GIFTS paper discusses a programming crash-course that represents a departure from traditional introductory programming education, designed specifically to capture the imagination of first-year engineering students through tangible, real-world applications. Early exposure to both programming fundamentals and programmatic thinking can set students up for later academic success, but often these concepts are presented in ways that do not resonate with students. Rather than confining students to abstract desktop programming exercises, the course introduces programming concepts through physical computing experiences that bridge the gap between code and the physical world.

Introduction

Programming and programmatic thinking are skills that benefit engineers of all disciplines, but courses designed to introduce these concepts are often meant to be a first-step into large-scale software development rather than pragmatic scripting or computation for engineers and scientists. Additionally, students who don't envision themselves programming heavily in their careers can have trouble linking the concepts they might learn in these types of courses to concrete results in the "real world" when the majority of their code written in such courses runs in a terminal on a personal computer. The course described in this paper was designed to give engineering students an engaging introduction to programming and programmatic thinking geared around instant feedback from a programmable hardware device–namely, the Adafruit Circuit Playground Express [1]. This device can be configured to run code that is written in several environments. However, for this course, we installed the supported CircuitPython interpreter on the device's ARM Cortex M0 microcontroller, and then used the CircuitPython [2] subset of the Python programming language to introduce programming concepts through weekly challenges that interfaced with the device's numerous sensors and smart LEDs.

This course was originally offered as a Special Topics class in a week-long workshop in Spring 2019. It was added to the core engineering curriculum at Authors' University starting in Spring 2021 and offered continuously as described in the present work through Spring 2024.

Alternate Approaches to Programmatic Thinking

Learning objectives in introductory programming courses typically extend beyond language-specific proficiency. Introducing and reinforcing algorithmic and programmatic thinking are often major components assessed in these types of courses as well. To that end, there have been several reports of alternate strategies to introductory programming that lead with procedural pseudo-code generation before any programming environment is introduced [3], [4]. We similarly emphasize programmatic thinking over syntax, except that we push students into the programming environment from day one, while also encouraging them to adopt an experimental approach to programming through modification of existing code.

Comparison Between Present Work and Traditional Programming Course Structure

In this section, we will compare a traditional approach to teaching standard programming concepts to the approach that emerged in our class. In general, the natural flow of the class is in some ways flipped. Rather than starting with syntax, variables, types, program flow, loops, and functions, then moving to practical implementations of programs, this course typically starts with a working script for motivation, then focuses on modifications to that script to reinforce concepts. The Mini Projects referenced in this section are provided in Appendix A.

Datatyping and Strings

Traditionally, datatyping and strings are introduced at the beginning of the course since datatyping is a requirement for strictly typed languages like C or C++. The programming assignment would require the students to write a simple program that declares variables and prints the variable to the standard output. Though datatyping is not always required in Python, typecasting is introduced as a means to force the type of a variable. Likewise, the student can observe how various types are rendered on the standard output.

In the present work, variables are most often used to store sensor values. For example, Mini Project 1 required students to use the signal from a light intensity sensor to infer the distance of the sensor from a flashlight. This forced students to engage with the concept of integer vs floating point types when interpreting the sensor values. Strings were introduced early on through the use of print statements in the debugging process. Tuples, lists, and dictionaries were introduced to create a pentatonic piano in Mini Project 5.

Loops

In a traditional programming class, loops are often first introduced by printing a sequence of numbers (the Fibonacci Sequence is common). Looping through lists and strings are additional common examples.

In our class, loops are used from week 1 on because in most embedded systems, code is placed inside of an infinite while loop. Both Mini Projects 5 and 8 use loops to iterate through a sequence of notes and encrypted characters, respectively. These activities explicitly introduce students to the concept of iteration through lists.

If Statements

Program flow is introduced in introductory programming courses almost exclusively by comparing characters or numbers to ranges or thresholds. While Boolean conditional statements always perform comparisons like these, the sources of information in the present work are varied and can be more tactile and captivating than text entered on a keyboard.

In Mini Project 3, students were tasked with comparing the output of their board's temperature or brightness sensor to a range of values, and lighting up LEDs on the board accordingly. Mini Project 5 asks students to use if statements to interpret pushed buttons for the purpose of playing one of eight different notes on the board's built-in speaker.

Functions

In a traditional class, it can sometimes be difficult to create situations where functions actually simplify the end product. Functions can be demonstrated in very simple environments, but in those cases, the stated goals of functions (encapsulation, abstraction, and code reuse) are very rarely realized.

In the present work, Mini Project 4 requires students to use functions to encapsulate parts of an LED light show, such as fading brightness and automatically transitioning between colors. These functions could be triggered by button presses or other inputs. In Mini Project 5, students were tasked with writing a function that would play a song through their board's built-in speaker. And in Mini Project 8, students wrote a function to encrypt or decrypt a text-based message.

Data Structures

In a traditional setting, various means of storing data tend to be very abstract. In a C++ course, you may introduce arrays. A programming assignment could involve generating random numbers, storing them in an array, and computing statistics on the data.

In this class, dictionaries are introduced in Mini Project 5 through the design of a pentatonic piano. Note frequencies are stored in a dictionary for more natural songwriting. Lists are introduced as a means of playing each of those notes, with time values attached for the duration of the notes. In Mini Project 6, a list structure was used to sequence a randomly generated secret code.

File Input/Output

In a traditional setting, file input and output may be introduced as an alternative to using the standard command console. An simple assignment may be to read data from a file, edit the data, and store it in another file.

In this class, Mini Project 9 introduced students to the use of files for storing settings and data. In engineering, data logging is frequently used to record sensor data. Students learn about data files by logging accelerometer data in CSV format, which can be opened in spreadsheets for further data analysis.

Why Use Python?

Since Python is a scripted language, the Python code is executed by the CircuitPython interpreter at runtime. In languages like C and C++, the code is compiled before runtime and any syntax errors must be fixed prior to execution. By using an interpreted language, like Python, the code immediately executes and allows for partially working code.

In addition, Python has become the most widely used language, by certain metrics[5]. It is generally considered easy to learn, while also being adaptable to a wide range of use cases and industries.

Student and Faculty Feedback

While end-of-course evaluations are known to be flawed, they can provide insight into the aspects of a course that most and least resonate with students[6, 7]. Here, we reflect on and respond to a few constructive end-of-course comments from students in this introductory programming experience.

Student feedback:

- "maybe some programming that we could use on a rasberry pi [sic] or something" and "[...I] would recommend more projects that did not involve the CircuitPlayground just to get more variety in the projects."

Broader applicability is a good goal. While the language students learned in this course was Python, it's easy to see how a student could perceive the scope of their programming ability to be limited by the hardware we used. This could be addressed by adding simulation in a desktop environment.

- "Possibly more engaging assignments? Some were kind of copy the code and change it barely while others were genuinely very interesting"

This is a legitimate problem with several of the current assignments. There is some complexity involved with including the API for the board, so the things actually being changed by the students can feel small in comparison.

- "I loved this class even though I am not a super big fun of program [sic]."

Ideally, the class would provide an onboarding experience to programming that makes it feel available to all of our students, but at the very least, providing a positive experience related to programming can be valuable for future learning.

Conclusions

Overall, this method of structuring the introductory programming course was effective at teaching programmatic thinking and basic programming skills. We would recommend it for students who are not Computer Science majors, whose desired skillset is focused more on developing small products that work, rather than building small pieces of large programming projects.

The biggest advantage of this paradigm was the immediate tactile feedback provided by the CircuitPlayground board. The output of the programs that the students wrote was a sequence of lights or a song, instead of text on a screen. This gave a level of intrinsic motivation in the mini-projects that is difficult to achieve in a traditional programming class. Additionally, the focus shifted to practical problem solving in programming, rather than large-scale programming paradigms which are less useful for the mechanical and civil engineering students who were involved in the class.

The biggest disadvantage was that students were required to use "black-box" statements from Adafruit to control the board. There is some silver lining to this, as effectively using black boxes is an important programming skill. In addition, the class in its current form did not include instruction on generating plots or solving equations, which are useful skills in following courses.

References

[1] lady ada and Kattni Rembor, “Adafruit Circuit Playground Express,” learn.adafruit.com. https://learn.adafruit.com/adafruit-circuit-playground-express (accessed Jan. 15, 2026).

[2] lady ada, Jay Doscher, and Michael Schroeder, “Welcome to CircuitPython!” learn.adafruit.com. https://learn.adafruit.com/welcome-to-circuitpython (accessed Jan. 15, 2026).

[3] Davies, S. 2008. “The effects of emphasizing computational thinking in an introductory programming course.” 2008 38th Annual Frontiers in Education Conference. T2C-3–T2C-8. DOI: 10.1109/FIE.2008.4720362

[4] Shrestha, A., R. Ishikawa, M. Sano, and S. Ooi. 2024. “Research on Support System for Programmatic Thinking Based on Metacognition of Instructional Interaction.” 2024 12th International Conference on Information and Education Technology (ICIET). pp. 253-257. DOI: 10.1109/ICIET60671.2024.10542708

[5] Cass, S., “The Top Programming Languages 2025,” spectrum.ieee.org. https://spectrum.ieee.org/top-programming-languages-2025 (accessed Feb. 17, 2026)

[6] Arthur, L. 2009. “From Performativity to Professionalism: Lecturers’ Responses to Student Feedback.” Teaching in Higher Education 14 (4): 441–454.

[7] Ballantyne, R., J. Borthwick, and J. Packer. 2000. “Beyond Student Evaluation of Teaching.” Assessment & Evaluation in Higher Education 25 (3): 221–236.

Appendix A - List of Classroom Activities

Mini Project 1: Distance Detector

Using the light sensor on your Circuit Playground Bluefruit (CPB), measure its distance from a fixed light source.

On the CPB, the light sensor measures different intensities of light. By taking measurements, you can regress the relationship between the light sensor value and the distance from a known light source. Using an equation that you will derive, you can then compute distance from light sensor values.

https://learn.adafruit.com/circuitpython-made-easy-on-circuit-playground-express/light

https://learn.adafruit.com/adafruit-circuit-playground-bluefruit/playground-light-sensor

Mini Project 2: Message Printer

Using the capacitive touch pads and push buttons on the Circuit Playground Bluefruit (CPB), print messages to the console. Be creative and have fun with your messages!

You will need to check the capacitive touch pins and the push buttons inside of an infinite loop. If a button is pressed, print out a message of your choosing. You should have at least 8 different messages.

https://learn.adafruit.com/circuitpython-made-easy-on-circuit-playground-express/capacitive-touch

https://learn.adafruit.com/adafruit-circuit-playground-bluefruit/circuitpython-cap-touch

https://learn.adafruit.com/circuitpython-made-easy-on-circuit-playground-express/buttons

https://learn.adafruit.com/adafruit-circuit-playground-bluefruit/circuitpython-digital-in-out

Mini Project 3: Temperature or Brightness Indicator

Make the NeoPixel LEDs on your Circuit Playground Bluefruit (CPB) indicate values coming from the light sensor or temperature sensor! This is a choose your own adventure!

Option 1: Temperature Sensor

A temperature range should be selected that is between a minimum and a maximum temperature. When the temperature sensor is within the selected temperature range, the color of the NeoPixel LEDs should indicate the measured temperature. When the temperature sensor is outside of the selected temperature range, the NeoPixel LEDs should blink. An appropriate range should be selected for demonstration purposes. The NeoPixel LEDs should change between at least 2 colors (ie. red for hot and blue for cold).

Option 2: Light Sensor

A light intensity range should be selected that is between a minimum and a maximum value. When the temperature sensor is within the selected light intensity range, the color of the NeoPixel LEDs should indicate the measured light intensity. When the light sensor is outside of the selected temperature range, the NeoPixel LEDs should blink. An appropriate range should be selected for demonstration purposes. The NeoPixel LEDs should change between at least 2 colors (ie. white for darkest and blue for brightest).

https://learn.adafruit.com/circuitpython-made-easy-on-circuit-playground-express/light

https://learn.adafruit.com/adafruit-circuit-playground-bluefruit/playground-light-sensor

https://learn.adafruit.com/circuitpython-made-easy-on-circuit-playground-express/temperatureLinks

https://learn.adafruit.com/adafruit-circuit-playground-bluefruit/playground-temperature

https://learn.adafruit.com/circuitpython-made-easy-on-circuit-playground-express/neopixels

https://learn.adafruit.com/adafruit-circuit-playground-bluefruit/circuitpython-neopixel

Mini Project 4: Wildcat LEDs

Make the NeoPixel LEDs on your Circuit Playground Bluefruit (CPB) display a moving purple, white, and rainbow pattern!

The LEDs on your board should do the following, in this order:

1) The brightness of all lights slowly increases from completely off to maximum brightness. Their color is white.

2) The color of every other light changes gradually from white to purple.

3) The lights that remained white blink through the following colors: green, blue, yellow, magenta, cyan, red, blue.

4) The brightness of all lights slowly decreases from maximum intensity to completely off.

Each of these steps should be encapsulated into a function. Inside those functions, you will need a for loop. Red, green, and blue are easy to display, but yellow is red and green combined, cyan is green and blue combined, and magenta is blue and red combined.

https://learn.adafruit.com/circuitpython-made-easy-on-circuit-playground-express/neopixels

https://learn.adafruit.com/adafruit-circuit-playground-bluefruit/circuitpython-neopixel

Mini Project 5: Pentatonic Piano

The CPB has a small speaker that can play notes! This week, we're going to build a pentatonic piano with the CPB.

Here's how the pentatonic piano will work. When you press a capacitive touch pad (A1-A6), a note from a pentatonic scale will be played. The note should be played continuously while the capacitive touch pad is being touched. The Neopixels should display a color and/or pattern indicating the note being played. When you press button A, a song should be automatically played using the same pentatonic scale. When you press button B, a song should be automatically played using the same pentatonic scale. While the song is playing, colors and/or patterns should be displayed.

https://pages.mtu.edu/~suits/notefreqs.html

https://www.pianolessonsontheweb.com/blog/take-5-and-learn-about-the-pentatonic-scale

https://learn.adafruit.com/circuitpython-made-easy-on-circuit-playground-express/capacitive-touch

https://learn.adafruit.com/circuitpython-made-easy-on-circuit-playground-express/start-and-stop-tone

https://learn.adafruit.com/circuitpython-made-easy-on-circuit-playground-express/play-tone

https://learn.adafruit.com/circuitpython-made-easy-on-circuit-playground-express/buttons

https://learn.adafruit.com/circuitpython-made-easy-on-circuit-playground-express/neopixels

Mini Project 6: Code Breaker Game

This week, we're going to build a simple game with the CPB. In this game, you will need to decipher a secret code on your CPB by pressing the two pushbuttons on top of the board in the correct sequence.

Here's how the game will work. Your board will generate a secret sequence of 10 button presses (button A or B) that you will try to guess. As you press one of the two buttons on top of your board, the Neo-Pixels will light up, one-by-one, with each correct button press. If you press the wrong button, your board's LEDs will all flash red twice, and it will then clear all the Neo-Pixels, indicating that you should try again. If you make all 10 button presses correctly, after the last Neo-Pixel on your board lights up, the Neo-Pixels board will cycle through multiple colors rapidly, and then a new code will be generated, the Neo-Pixels will reset, and the board will be ready for you to start guessing the new secret pattern again. A new pattern will only be generated if you win, so if you fail to enter the correct sequence of presses, you'll be able to use your progress from before you made a wrong press to get further into the correct sequence.

https://circuitpython.readthedocs.io/en/2.x/shared-bindings/random/__init__.html

https://learn.adafruit.com/circuitpython-made-easy-on-circuit-playground-express/buttons

https://learn.adafruit.com/circuitpython-made-easy-on-circuit-playground-express/neopixels

Mini Project 7: Remote Control Tone Generator

The CPB can communicate using Bluetooth Low Energy (BLE)! Today you'll make your board play different tones by controlling it remotely.

Here is how the Remote-Controlled Tone Generator will work. You will pair up with another student to complete this project. One student will use their CPB as the remote and the other students will use their PCB as the receiver. The remote will use 8 different buttons (button_a, button_b, touch_A1, touch_A2, etc.) to send bluetooth messages to the receiver. The receiver will receive those messages and play 8 different notes.

Each board has a BLE transceiver, and we'll be using a python library that communicates using a UART Service over a BLE communication link. You will need to pair with another student. With many students near each other, your board may accidentally pair with boards other than your partner. If the boards are close to each other, they are more likely to pair.

The necessary libraries should already be on the board in the lib folder. There should be a folder called adafruit_ble in the lib folder on the board. If not, the BLE library can be found hereLinks to an external site.. Extract the zip file on your computer, and in the lib folder find the adafruit_ble library folder. You'll need to drag that file into the lib folder on your Circuit Playground Bluefruit.

Before you can start making tones play, you will need to get a BLE connection between the two of the Circuit Playground Bluefruit boards. You and your partner will need to decide upon which messages will be sent.

Next, you'll need to refer back to the example from last week about playing notes for a certain amount of time. If you receive a BLE message and IF it matches one of your messages, then play the desired note for that button press.

Working BLE example code: https://courses.ideate.cmu.edu/16-376/s2021/ref/text/code/cpb-ble-sensor.html

The BLE tutorial: https://learn.adafruit.com/adafruit-circuit-playground-bluefruit/getting-started-with-ble-and-circuitpython

The play tone tutorial here: https://learn.adafruit.com/circuitpython-made-easy-on-circuit-playground-express/play-tone

Mini Project 8: Encrypted Messages

Today you'll make your board send and receive encrypted messages. You will be using the Adafruit BLE Library. Since this project will require two boards, it will be completed with a partner.

Here is how the encrypted messaging device will work. The device will use a rotational cypher, which is also known as a Caesar Cypher, for both encrypting and decrypting messages. The user will set the rotational amount for the cypher using buttons A and B. The amount of rotation being used for the encryption will be indicated on the Neopixel LEDs. The messages being transmitted will only contain capital letters (A-Z).

When sending, the capacitive touch pads select which message being sent. Since there are 7 capacitive touch pads, the device should be able to send 7 different messages. When the capacitive touch pad is touched, the encrypted message should be sent.

When receiving, the device will store the encrypted message. Using the rotational amount setting, the encrypted message will be decrypted and the decrypted message will be printed to the serial terminal.

https://en.wikipedia.org/wiki/Caesar_cipher

Mini Project 9: Kinetic Data Logging

Today, we'll use the Circuit Playground Bluefruit to log data from its accelerometer into a data file. Here is how the acceleration data logger will work. There will be a switch that toggles read/write access to the storage on the CPB. In one position, the data file and code will be accessible by the computer. In the other position, the code will write acceleration data to the data file. The device must somehow indicate that it is datalogging. The device must somehow indicate the most recently measured acceleration.

Accelerometers always sense gravity. If the CPB in sitting on a table, it will measure gravity on its z axis.Here is code for accessing the CPB storage. You will need to create boot.py file.

https://learn.adafruit.com/adafruit-circuit-playground-express/circuitpython-storage

https://learn.adafruit.com/circuitpython-made-easy-on-circuit-playground-express/acceleration