INTODUCTION TO ARDUINO CODING

Peter Vaughan

NSCC ONLINE CAMPUS Self-Study

Table of Contents

Teaching Electronic Topics

Objective: Design a 15-week class on how you would teach programming concepts to beginners in college or an undergrade program. Setup a mock curriculum on how you would teach beginners or intermediate learners on how to code. There is a long list of potential concepts and ways to teach how to code. Also, there are many other skills to learn while learning how to code, such as how to use GitHub and IDEs. There are many IDEs with all different capabilities or toolboxes on how the teacher or student would approach coding.

Then, consider how to mark the student if you were the teacher? This is a 15-week class. Assigning tasks too soon might lead to confusion or assigning task too late might not give students enough time to finish. Next, explain your logic on why you approach the class the way you did.

Basic Concepts

1. **Introduction to Programming**

- What is coding?
- Why learn to code?
- Overview of common languages

2. **Programming Basics**

- Syntax and semantics
- Variables and data types (Strings, integers, floats, Booleans)
- Operators (Arithmetic, comparison, logical)

3. **Control Flow**

- Conditional statements (`if`, `else`, `elif`)
- Loops (`for`, `while`)
- Loop control (`break`, `continue`)

4. **Data Structures (Optional for Beginners)**

- Lists/arrays
- Tuples
- Dictionaries/maps
- Sets
- Basic operations: indexing, slicing, appending, removing, iterating

5. **Functions**

- Defining functions
- Parameters and return values
- Scope (local vs global)
- Built-in vs. user-defined functions

6. **Error Handling**

- Syntax errors vs runtime errors
- Debugging basics
- Using (`try`, `except`)

7. **Input and Output**

- Printing to the console
- User input (`input()` in a popular computer language)
- Reading from and writing to files

8. Object-Oriented Programming (Optional for Beginners)

- Classes and objects
- Attributes and methods
- Encapsulation

9. Simple Projects

- Calculator
- To-do list
- Number guessing game
- Basic web scraper

10. Good Coding Practices

- Code readability
- Comments and documentation
- Naming conventions
- Version control (Intro to Git)

Data Structures and Object-Oriented Programming (OOP) is great to optimize programs. However, some of these concepts rely on a strong programming foundation depending on how these topics are approached. Beginners can be taught if the teacher adequately teaches the potential issues and are given potential examples on issues through tangible code. Vague answers and lack of proper examples may lead to issues.

I have crashed my computer with one line of code with data structures due to memory leakage in CLion IDE. CLion IDE a code compiler for C++ with many tools. It took me awhile to find the root cause of this memory management issue with data structures. However, with the Arduino and its IDE, the Arduino will not operate correctly. Results may vary. Data structures are vital to programmers but not usually beginners with basic programs.

If we did a second Arduino class, I would add more hardware components and teach memory management through data structure because program would typically be more complicated for the Arduino's microprocessor to handle. With my self-study program, I did my study on the Arduino code. My main issue was how I was managing data with the microprocessor when the program got to 1,000+ lines of code.

OOP can be broken down to four concepts: Abstraction, Encapsulation, Inheritance and Polymorphism. It is a wonderful way to write code if the student understands on how to best approach writing code this way. Simple examples are great for learning OOP. It takes some students some time to completely get this concept because it is not how most people think.

Potential Teaching Ideas with IDEs

Use Real-World Analogies

Explain an IDE as a “Toolbox for Coders”. Compare an IDE to Microsoft Word for writing essays—it helps you write, edit, and organize your code more easily, just like Word helps format documents. There are many other real-world analogies.

Hands-On Demo (Start with a Simple IDE)

Consider using beginner-friendly IDEs like:

- Thonny (Python)
- Replit (online, no installation)
- VS Code

Then show key features when programming with an IDE:

- Code editor
- Run button
- Output console
- Error highlighting

Let students type print(“Hello, world!”) and run it in a loop – let’s see the results.

Highlight Key Features One at a Time

Break each concept down into four components:

- **Syntax highlighting**: colors help identify parts of the code
- **Auto-complete**: faster typing, fewer errors
- **Error hints**: red underlines or warnings
- **Run & Debug buttons**: running code with one click

Let them explore by changing parts and watching the output or errors. A great programmer knows how to troubleshoot code.

Mini-IDE Scavenger Hunt

Create a small challenge such as “Find where you can run your code, see errors, and write comments.” This encourages exploration and familiarity. End with a small task such as “Write a program that asks your name and greets you.” There are so many examples and tutorials online.

Class Learning Outcomes and Submission

Class Learning Outcomes

1. Code Functionality

- Does the program work as intended?
- Are the outputs correct for given inputs?
- Are all required features implemented?

2. Logic and Problem-Solving

- Is the solution logically sound?
- Does the student show understanding of conditionals, loops, etc.?
- Is the code efficient (appropriate use of structures)?

3. Code Readability and Style

- Proper indentation and formatting
- Clear variable names
- Comments where appropriate
- Avoiding unnecessarily complex solutions

4. Debugging and Error Handling

- Can the student identify and fix errors?
- Do they use try/except or similar constructs (if taught)?
- Do they show understanding of common error types?

5. Documentation or Explanation

- Can the student explain their code?
- Do they include comments or a short write-up?
    - Include a ReadMe File.
- In an oral exam(s): Can they walk through the logic clearly?

Bonus or Extra Credit (Optional)

- Creative additions beyond the task
- Optimization or good use of advanced features (if optional)
- Clean user interface (for projects with GUI)
- Assessment of project and strategies

Submission

Submission for assignment(s) and project(s):

Emailed to teacher or submitted on Brightspace:

- Documentation
- Link to student's GitHub Repo
  a) Code
  b) ReadMe File
- Feedback Form – to create better feedback on the course

When I create assignments, I will simplify these coding concepts into four basic concepts: Learn basic coding, basic troubleshooting, interfacing with hardware, and documentation. My purpose of teaching is to encourage students to learn as much as possible in the way they think. This is a beginner class and I should not complicate what should be simple. Also, I will provide working example code. While students need to be well-rounded in terms of what they learn, I do not want to discourage them and I will provide constructive feedback on how they can improve.

Arduino Coding Class

TEXTBOOK / RESOURCE REQUIREMENTS

No textbook is required.

Laptop or desktop with:
- Windows 10 or Later, MacOS or certain Linux distributions
- 1 GB of hard drive Space
- 1.6 GHz or faster processor
- 1 GB of RAM

SUPPLIES / ADDITIONAL RESOURCES

1. Arduino Uno
2. USB 2.0 A Male to B Male Cord USB A to B Cable
3. 10 1k Ohm Resistors
4. 10 LEDs (any colors)
5. 30 Jumper Cables
6. Electronic Breadboard

Alternatives can be arranged if supplies are unavailable. The school should have some available for this purpose. Otherwise, we will look into a simulator.

Course Learning Outcomes

The four main key elements to develop coding for beginners and starting to explore embedded systems through the Arduino by learning these four concepts:

1) Learn basic coding
2) Basic troubleshooting
3) Interfacing with hardware
4) Documentation

Evaluation Scheme

Assignments: 6 @ 10% each	60%
Project: 1 @ 30% each	30%
Self-Study: 1 @ 10% each	10%
Total	**100%**

Schedule

Weeks / Unit	Topic / Descriptions	Relevant Learning Outcome(s)	Value/Evaluation/ Due Dates (if applicable)
Week 1	• Introduction to Programming: Understanding the IDE (VS Code)	1	
Week 2	• Programming Basics	1	Assignment 1 (10%) Week 3
Week 3	• Control Flow • Understanding Errors • Assignment 1 Due	1, 2	
Week 4	• Functions • Error Handling	1, 2	Assignment 2 (10%) Week 5
Week 5	• Input & Output • Error Handling • Assignment 2 Due	1, 2	Assignment 3(10%) Week 6
Week 6	• How to Prepare Documentation and ReadMe Files • Learn about GitHub • Assignment 3 Due	4	Assignment 4(10%) Week 7
Week 7	• Getting Started with the Arduino and its IDE • Assignment 4 Due	1, 3	Assignment 5 (10%) Week 8
Week 8	• Basic coding with the Arduino • Understanding Basic Errors and Using Proper Error Handling • Assignment 5 Due	1, 2, 3	Assignment 6 (10%) Week 9
Week 9	• Simple Circuit Design • Code Readability and Style • Assignment 6 Due	1, 3	Self-Study (10%) Week 11
Week 10	• Share Basic Code for Mini-Projects for demos, troubleshooting and documentation	1, 2, 3, 4	
Week 11	• Share Basic Code for Mini-Projects for demos, troubleshooting and documentation • Self-Study Due	1, 2, 3, 4	Project (30%) Week 15
Week 12	• Good Coding Practices	1, 2, 3	
Week 13	• Working on project – Open to questions to explore the Arduino / embedded systems.		
Week 14	• Working on project - Open to questions to explore the Arduino / embedded systems.		
Week 15	• Project Due • Course Closure		

Additional Information

Inclusion and Integrity of the Learning Environment
We strive to ensure that equity, inclusion, and social justice are the reality for all students, faculty, and staff. We commit to providing a safe and respectful working and learning environment where differences are valued, expected and honoured. Within this environment, students are required to demonstrate the values of respect, academic integrity and honesty.

To support these goals, we have the following policies:

- Respectful Community
- Student Code of Conduct
- Employee Code of Conduct
- Sexual Violence
- Academic Integrity
- Academic Accommodations
- Educational Equity

Appealing a Final Grade

NSCC is committed to a fair, transparent and timely approach to a student's right to challenge academic decisions and non-academic decisions that affect academic progress and standing.

If you feel your final grade is unreasonable, speak with your faculty or Academic Chair about your concerns. If the issue is not resolved, you may pursue a formal appeal. Speak with your Student Services Advisor for more information on the Student Appeals policy, procedures, and your eligibility.

Copyright

Copyright compliance is a legal responsibility. All students, staff and faculty at NSCC are required to abide by the NSCC: Use of Copyright Materials Policy, Fair Dealing Guidelines and the Copyright Act of Canada when copying materials. This includes art, music, videos, sound recordings, images, printed works (book, journals, newspapers, etc.) and materials on the Internet. Check with your Campus Library if you have questions or visit our Copyright Guide.

Preparing for Learning

Your success in this course stems largely from your level of engagement and willingness to learn. Preparation, attendance, and participation are key factors in learning. If you feel overwhelmed, lost, or disengaged, speak with your faculty, Academic Chair, or Student Services Advisor about how we can help.

Student Supports

Student Services provides you with a wide range of supports. For more information, visit nscc.ca/services. For support with Brightspace contact the Technology Service Desk by visiting servicedesk.nscc.ca. Click Create a Request (Select “Brightspace (D2L)”, then "Brightspace (D2L) Student Support”). Or, by phone, dial 902 491-6774 (press 4), or Toll-free:1 877 491-6774 (press 4). For self-directed, how-to resources to aid in using Brightspace, visit the Brightspace (D2L) Toolkit.

Key Links

College Regulations: nscc.ca/Admissions/college_regulations.asp
NSCC Policies and Procedures: nscc.ca/about_nscc/publications/policies_procedures/index.asp

My Logic of Class Approach

This class is about learning about the Arduino. I have experience with leaning about embedded systems in post-secondary schooling. This was with hardware design programs. The teachers taught the hardware perspective but barely covered the coding. The coding was primarily self-taught in these programs. However, I am a strong believer that before we should learn about building embedded systems, we should learn how to code. Without the code, we cannot fully test the embedded system or verify its full functionality. It is then we can code a simple pre-made system like the Arduino with some additional components. After that, we can start assembling different embedded systems.

I went back to school for computer science – primarily for programming. With my new programming knowledge, I think programming a pre-made embedded system is the easiest way to approach teaching embedded systems. If I had a second class that continued these concepts, I would teach about the hardware, such as schematics and other components. Then I would encourage students to build simple embedded systems. We should learn embedded system from a coding perspective than a hardware perspective. It makes the learning process easier.

It was difficult for my classmates and I to build embedded systems in the hardware programs without programming knowledge. This is why I approached this introduction to Arduino class this particular way. For this introduction class, I would give pictures of how all the components are hooked up along with the schematic. The schematic might not mean much to some students but I want to encourage students to explore what concepts are out there. Students can surprise teachers with what they are willing to learn.

Additional Info

GitHub Repo: https://github.com/Vaughan-Peter/ArduinoLearning

Contact Info: https://www.linkedin.com/in/peter-vaughan-997478239/

Contact E-mail: otherhalifaxprojects@gmail.com