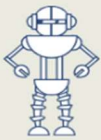




# DeyGana Project Based Engineering: CODING CURRICULUM

Unknown Unknowns to Known Unknowns to  
Known Knowns: The Ultimate STEM/STEAM  
Progression to Achieve Proficiency

Created by DeyGana



## Table of Contents

<b>DeyGana STEAM</b>	<b>4</b>
Vision	4
Curriculum Purpose	4
What is STEAM & Why is it Important?	6
Why Project Based?	6
Curriculum Overview	7
Scalability	8
Mentorship	9
Curriculum Topics	10
Coding/Programming	10
Video Game Development through Unity	10
Computer-Aided Design (CAD)	10
Mechatronics	11
Robotics	11
<b>Coding with Blocks, Hybrids, and Text IDEs using Code.org, HTML5, and Python</b>	<b>12</b>
Purpose	12
Learning Methods	12
Lessons: Code.org	15
Summary of Code.org Course	15
Course 1: Express Course	15
Course 2: Javascript-based App Lab: CSP Unit 3-7, 9	16
Course 3: Web Lab: Web Development	17
Lessons: Python	19
Summary of Python Course	19
Lesson 1: Express Course -> Sequencing, Loops, and Conditionals	19
Lesson 2: Introduction to Integrated Development Environments (IDE's), Commands, Data Types, and Variables	20
Lesson 3: User inputs, casting, basic logic, and pseudocode	22
Lesson 4: Conditionals and Comparators	23
Lesson 5: Boolean/Logical Operators	24
Lesson 6: Nested Conditionals	26
Lesson 7: While Loops	28
Lesson 8: Libraries and Arrays/Lists	30
Lesson 9: For Loops	31
Lesson 10: Putting it all Together with Rock Paper Scissors	33
Lesson 11: Functions	33
Lesson 12: Classes	34
<b>Beyond the Lessons</b>	<b>36</b>
Future Development	36
Additional Tips for Instructor	36

# DeyGana STEAM

This isn't about building, it's about learning how to build

This isn't about building a good boat, it's about learning how to build a better boat

## Vision

Teach students general knowledge within a particular subject matter and then place them within an environment to further develop those skills and discover what they don't know. They will then ask for answers about what they don't know, and we'll guide them towards those answers so that they can return to the development environment and keep working at it until they have more questions. Once all of this is done and they present their final creations, we will talk about it in class as a group so that you can see where everybody is, how the successful people did it, and how the unsuccessful did it so that all students can glean lessons from both sets of students. It allows them to see not only their teacher's perspective and approach, but also their classmates' perspectives and approaches

**Unknown unknowns:** subject matter & skills students don't know exist

**Known unknowns:** subject matter & skills students know about, but lack experience in

**Known knowns:** subject matter & skills students know about and can confidently explain/demonstrate

To summarize, students start with unknown knowns, we teach them general principles and give them a test to turn those unknown unknowns to known unknowns. Then by addressing those known unknowns, they become known knowns and the students can return to the test to discover more known unknowns that we can continue to transform into known knowns. Finally, the reflection should address other unknown unknowns that the initial lesson never brought up. A group reflection allows multiple students to say and discover something that the teacher never discussed in the first place, and we can also turn those unknown unknowns to known unknowns and finally known knowns!

This process allows you to efficiently create personalized education for your students because after the initial lesson which introduces students to the subject matter and transforms unknown unknowns to known unknowns, each student will only come to you **with questions specific to their progress**. They will discover what they need at their own pace, you just need to provide them with an adequate test and time to struggle and develop.

## Curriculum Purpose

By the grace of God, we have been fortunate enough to build programs for multiple organizations across nations and continents that successfully and passionately pursue the goal of teaching students how to learn and providing them with resources to support their directions. The purpose of this curriculum is to introduce and expose students to state-of-the-art education methods through conversations, demonstrations, and challenges that will be fun, educational, and relevant. We aim to inspire students to take a deeper look into STEAM (Science, Technology, Engineering, Arts, Math) related activities, the humanities, finances, and everything else that contributes to holistic, caring, and action-oriented people.

In doing this, we understand that certain schools have resources that other schools across the globe don't have even in high GDP nations. We believe that while this plays a large role in why many of the affected schools don't have STEAM programs, it shouldn't. There is a lot of fat i.e content or hardware that is not critical in the initial stages, in many STEAM programs when it comes to equipment and curriculum. Upon growth and reflection, we noticed this within earlier iterations of our own program. As a result, we've found different tools to replace what we used to believe was necessary in the classroom. \$500 robots have turned into free online simulators and \$35 electronics kits. Lack of space for computers in the room to program microcontrollers has turned into virtual circuits on laptops where every student can individually build and program at their own pace without getting lost in a group.

Our curriculum aims to trim and simplify curricula to provide foundational essentials for the understanding of our students. We have seen teachers waste time training students who don't have the math and science foundation to understand HOW things work even though these same students don't need to understand how things work in order to use them just like you don't need to understand how a motherboard works to use a computer or how a car engine works in order to drive one. Some of our students cannot tell you the first thing about how an ATmega328 chip works or about flip-flops or motor controller functionality but they can use those same devices to build an electric skateboard with a brushless dc motor, a motor controller, and an Arduino to control some cool LED lights they attached to the bottom.

This revelation showed us that while technical knowledge and the *how* is good information to have around you, it only becomes useful when it is relevant to the task at hand. If not, then it's just extra information that the brain will soon purge. This is the reason why our classes are based on introducing students to a range of topics on a surface level just for exposure first; afterwards, we assign them projects that fall in line with their interests which will force them to learn the technical information and theory within their subject matter because it becomes relevant to them completing their projects. **It no longer becomes about learning information just for the sake of knowing it or passing a test. It becomes about learning the information in order to immediately apply it in attempts to craft a USEFUL solution to chosen project/problem statements.** We can continue to list more and more activities; however, we have demonstrated our point with these examples.

Not only will students be able to get a high quality STEAM education that will build their creativity, problem-solving skills, and technical abilities within coding, manufacturing, robotics, and other engineering, humanities, and financial spheres and beyond, but also they will be able to do it at a fraction of the cost of your average STEAM program in a **RELEVANT** way.

**Our goal is to sever the relationship between a lack of resources/math and science foundation and a lack of a STEAM program.** While those things are currently related in the way we look at STEAM these days, we truly don't believe that they have to be. We believe engineering is intuitive enough that anyone with zero background in it can be able to look at what's going on and at the very least say "even though I don't understand the ins and outs of how it works, you pointing out the different elements and their functions within the system makes sense as to why you built it that way." It's too vital to divorce those ideas from each other with where the world is headed because students who don't have the opportunity to explore this kind of education will be left behind due to the excuses we're providing as engineers, educators, and adults responsible for their lives.

## What is STEAM & Why is it Important?

STEAM stands for Science, Technology, Engineering, Arts, and Mathematics. The point of a STEAM education is to complement a classical education in a developing world. While learning about theory is important, developing a problem-solving mindset is particularly important in this day and age to building a holistic citizen. A STEAM education places emphasis on subject matter that promotes this kind of logical and rational problem solving while collaborating with the arts in order to introduce creative and non-traditional elements to the process in order to produce truly limitless solvers of problems of the past, today, and beyond.

## Why Project Based?

People tend to look at projects as a presentation/culmination of your knowledge on a subject. We actually look at projects as lectures, lessons, and teaching tools. It just so happens to be that the student is directing HOW they're going to learn their topic of interest. **E.g., a student is learning web development by embarking on a project to create a personal blog site that has at least x amount of pages and y amount of dynamic content.** This will involve all the HTML, JavaScript, CSS coding as well as learning about domains, hosting, servers, and SSL certificates all from a project about making a website.

A student is learning about Arduino powered mechatronics by embarking on a project that involves building and programming a robot hand as opposed to assembling one from a kit. This involves physically building the hand whether it's through CAD or hand built, sourcing

appropriate motors, motor controllers, and other electronics. If it's wireless, they would include a transmitter and receiver. If not, they would use a controller or sensors compatible with the Arduino.

It involves all the programming that goes into making sure all the elements are properly integrated. It also involves **1)** learning about the electrical components and making sure the proper power is supplied and there are no shorts, **2)** learning about resistors and capacitors and their role in circuits, **3)** learning about soldering, stripping, and connecting, and insulating wires, and **4)** learning about prototyping on breadboards before creating permanent circuits with PCBs, among other things involving iterating on designs, comparing products before purchase, wisely spending budgets, etc. To successfully complete this project, the student must learn all these things on the way there. Learning these skills on their own can be boring and random because there is no theme and mission but as soon as you create a project that connects and necessitates those skills, they suddenly become relevant and elevated material.

A student is learning about flight dynamics by doing a project which will result in them building and successfully flying an RC plane. This means learning about different wings, propellers, power supplies for the plane, aerodynamics and the shape of the fuselage, wind resistance, transmitters and receivers and their range, etc.

In our programs, students are not building their projects to show off and demonstrate skills and knowledge they already have. That would be a waste of time. Why do what you already know unless you have a good reason to? They're doing these projects because they're trying to build on their current foundations, acquire new skills, and hone the ones they already have and what better way to do that than by building a project that makes all the skills you're trying to learn a necessary part of completing the task at hand

## Curriculum Overview

These sessions will primarily be held for **3 different age groups: Elementary School (Grades 3-5), Middle School (Grades 6-8), and High School (Grades 9-12)** but can also be modified for varying experience groups of beginner, intermediate, and proficient as well. Each group will have its own set of stimulating activities and challenges depending on the date of the session they attend. While this is the general approach, please note that part of implementing the program is adjusting to the needs of students who either need more help or stronger challenges to properly engage with what is in front of them.

- **Grades 3-5:** The elementary group will typically be limited to computer education and robotics, easy assembly/pre-assembled materials, and craft work (popsicle sticks, LEGOs, straws, skewers, paper cups, and plates, etc.) that do not require much beyond tape, school glue, and safety scissors.

- **Grades 6-8:** The middle school group will typically be allowed to do more labor-intensive work on top of what the elementary group is already doing. They will also use simple craft items like popsicle sticks, and straws; however, they will be allowed access to more modification materials like hot glue guns, box cutters, soldering irons, etc. This group will also learn about coding logic and programming through different challenges.
- **Grades 9-12:** The oldest age group, the high school group, will have access to power tools like drills, power sanders, electric saws, epoxy, etc. Their challenges will also be more build intensive since they will have access to more building materials like 2x4s, plywood, PVC pipes, and other such materials. This group will also learn programming languages and those who are ready will have more intense programming challenges prepared for them.

There will be various challenges available depending on the skill levels of those participating. The point of these programs is to not hold back any students who are farther along in STEAM education than their peers. If this ever becomes the case, more advanced challenges should be provided.

The goal of these sessions is to provide students with access to the state-of-the-art in engineering and education, something that usually comes at a steep price, at a very low cost. This will allow students to not just read about engineering in textbooks or watch videos about it online, but experience it first-hand. It will open their world when they come to understand the possibilities that exist on a STEAM track. And finally, it may even give them an idea of the path they want to take in their future or reveal to them the path they do NOT want to take after spending time with our program allowing them to shift their focus and attention elsewhere.

## Scalability

This program is scalable and sustainable because once students get knowledge in certain areas, they can teach each other and help their struggling classmates get up to speed. Many do it better than their teachers because they speak the language of their peers. And for the students who pick up the subject matter quickly, you can give them more challenging versions of the same assignment with less abstraction that can start to test not only the technical skills they are learning, but also their logic, reasoning, and knowledge of WHEN to apply different principles which is when they start to stand on their own feet and develop independence. That's the goal here: independence. Not just knowing programming techniques, but when to apply them. Not just knowing CAD techniques, but when to apply them. Not just knowing how to use a jigsaw or drill or hammer, but when to use it. Once students prove their efficacy in this area, the program has given them all you can give.

The only thing that is left to do is to give them projects/problem statements within the areas they're interested in growing their knowledge and let them go. Try to be as invisible as possible

during this time. Do not be their first point of contact, second, or even third. If they don't have you there to lean on for answers, they'll start learning not only how to find answers on their own, but also how to find **RELEVANT, USEFUL, and APPLICABLE answers**. There is plenty of information on the internet, in books, and in other people. A lot of it is useful, a lot of it isn't. One of the most important skills this program can teach a student is how to differentiate between those two categories with regards to their goals. Just because something is useless today doesn't mean that it will still be useless tomorrow.

In these projects, they will develop depth in the skills that the program gave them surface level exposure in depth that cannot be simulated with a lesson or even an assignment. The reason is because of **RELEVANCE. That's the keyword here**. If the deepest level of relevance of any subject matter is attached to a grade, then the assignment is useless. Unless the student has their own agenda, the information will remain relevant to them until that grade has been submitted; however, if there is something outside of a grade attached to the subject matter like a problem statement/project that leans on the technical skills and knowledge in that subject matter, the student will have no choice but to constantly apply those technical skills and knowledge in that area until they solve the problem or come to the realization that the problem is beyond their current scope. With either of those results, the student either understands the problem because they've solved it or because they understand what it takes to solve it and why they're currently not in a position to do so.

## **Mentorship**

We will create time during this program to allow people who work in various STEAM industries (automotive, cyber security, aviation, etc.) to engage with and speak to our students. We believe that it can be extremely important to know the background behind who someone is, what they do, and how they journeyed to where they are. It will give students an idea of different paths that may be available, the challenges and pitfalls that come with those paths, and most importantly how they can potentially overcome them. We also believe that it is important for students to see that being an engineer or a scientist is not the only career available in the STEAM fields. You can become a pilot, an astronaut, a programmer, an architect, project manager, and the list goes on and on. We are excited to collaborate with passionate professionals to show our students how exciting these different career opportunities can be!



# Curriculum Topics

## Coding/Programming

- **Needs:**
  - Laptop
  - Chromebook
  - Tablet
  - Raspberry Pi 400 (this is the best option because it allows students to learn CAD, coding/programming, and embedded systems either using an arduino or the raspberry pi itself)
- **Source Material**
  - code.org for introducing coding concepts such as sequencing, loops, conditionals, javascript, app building, etc.
    - Code.org technical requirements - <https://support.code.org/hc/en-us/articles/202591743-Technical-requirements-for-Code-org>
  - Edublocks for introducing python syntax

## Video Game Development through Unity

- **Needs**
  - Laptop
  - Chromebook
- **Source Material**
  - Unity Website
  - Unreal Engine Website
  - Youtube
  - Code.org

## Computer-Aided Design (CAD)

- **Needs**
  - Laptop
  - Chromebook
  - Tablet
  - Parametric CAD software: SolidWorks, Onshape, Fusion360
  - Direct Modeling software: TinkerCAD, Blender
  - 2D Photo Editing or Vector Graphics Software: Inkscape, Adobe Illustrator, Corel Vector
- **Nice to Haves**
  - 3D Printer: Prusa, Ender, etc.
  - Laser Cutter
- **Source Material**
  - Youtube
  - Forums

- CAD Software websites and FAQs

## Mechatronics

- **Needs**
  - Laptop
  - Chromebook
  - Tablet
  - Mechatronics Simulator: TinkerCAD Circuits, micro:bit simulator, wowki
- **Nice to Haves**
  - Arduino Kit
  - micro:bit Kit
  - Raspberry Pi Kit
- **Source Material**
  - Youtube
  - Forums
  - Manufacturer Resources

## Robotics

- **Needs**
  - Laptop
  - Chromebook
  - Tablet
  - Robotics Simulator: VEX VR
- **Nice to Haves**
  - LEGO Mindstorms/Robot Inventors kits
  - mBot kits
  - VEX kits
- **Source Material**
  - Youtube
  - Forums
  - Manufacturer Resources

# Coding with Blocks, Hybrids, and Text IDEs using Code.org, HTML5, and Python

## Purpose

According to [Kenneth Dakin from Quora](#), "Coding is simply the process of providing instructions to a machine in a certain way. Since most home computers produce mainly printed output or visual output on a monitor screen, the instructions (your code) tells the machine how to transform your "input" (keystrokes for instance) into some other form - as its 'output.'" These are some reasons why learning coding is valuable:

1. Coding is used to create software, apps, and websites
2. It is also used to run embedded systems like microwaves, alarm clocks, televisions and their remotes, etc.
3. Apart from the job opportunities learning coding provides, it is also useful in that it challenges the learner's logic, critical thinking, and creativity thereby building their overall problem solving skills as it pertains to completing coding and programming challenges
4. Other intangible qualities coding also develops in learners is tenacity, perseverance, patience, and an astute attention to details.
5. Margaret Mead says "children must be taught HOW to think not WHAT to think." Learning coding provides an environment that teaches this because just like with evolution, attempts at solutions birthed from impractical thinking dies away while the thought processes that yield useful solutions rise to the top.
6. Coding also provides the opportunity for students to create solutions that do not require much capital since they are developed on a computer and can be shared with the rest of the world through the internet which is much cheaper than most physical solutions
7. Finally, coding also gives students the opportunity to apply content from their other classes in a real and practical way: it allows them to take lessons from text books into the real world.

This is why learning coding is important not only for kids interested in computers, but for kids in general. In a world where we have access to so much information in microseconds, the problem isn't the lack of information but rather the means of differentiating what information is useful, relevant, and correct, and what isn't. Not all answers are created equally. Not all answers can be applied equally. Learning how to think arms students with the skills necessary to separate the wheat from the chaff and an education in coding and programming provides just that and more.

## Learning Methods

There are two primary methods we will be using to learn coding. Block coding and Integrated Development Environments (IDE's) for written code. The main languages we will be dealing with are python and javascript; however, this applies to multiple programming languages. The resources for these methods are all web based and so anyone with a computer and an internet connection can have access to these programs. You can also download an IDE that is local to the machine.

- The first resource we will be using is [code.org](https://code.org) which is our primary resource for javascript based programming. The reason we use this course is because it allows students to learn the concepts of coding without having to deal with the syntax and details of a programming language. We will use this to introduce younger students to coding because it has a friendlier user interface. It is free to create user accounts and the program is webapp so all a student needs to run it is a computer with an internet connection. Code.org also gives teachers the ability to create virtual classrooms where they can create accounts for students, see their projects, see their exercises and provide solutions for them, and monitor their progress. Code.org also has resources for anyone who wants to be self-taught by way of lessons, follow along projects, and galleries of projects that users can attempt to remake or remix. According to [Career Foundry](https://www.careerfoundry.com), here is why you should learn Javascript:
  - Javascript's primary use is in developing webapps, websites, desktop apps, and game development. You can even build some of these elements from scratch
  - You can code on the client-side (frontend) using Angular and on the server-side (backend) using Node.js. You can also develop web, mobile, and desktop apps using React, React Native, and Electron, and you can even get involved in machine learning.
  - JavaScript frameworks contain sets of prewritten, ready-to-use JavaScript code. You can think of a framework as your blueprint for building a website: it gives you a structure to work from, and contains ready-made components and tools that help you to build certain elements much quicker than if you were to code them from scratch. Some popular JavaScript frameworks include Angular, React, Vue, and Node.js.
- The second set of resources we will be using for python is [EduBlocks](https://www.edublocks.org) and [Google's Colab](https://colab.research.google.com/). We will use EduBlocks, similarly to code.org, to introduce beginners to the concepts of coding because this will allow them to focus on the concepts without having to deal with the syntax and details of a programming language. EduBlocks and Colab are cloud-based webapps. What this means is that a user only needs an internet connection and an internet connected device like a phone, computer, or tablet to use it. According to [Code Institute](https://www.codeinstitute.io) and [Upgrad](https://www.upgrad.com), programmers can use Python for the following:
  - Automate tasks in webapps
  - Conduct scientific and mathematical computations

- Web Development
- Game Development
- Artificial Intelligence and Machine Learning
- Software Development
- Enterprise-level/Business Applications
- Education programs and training courses
- Language Development.
- Image Processing and Graphic Design Applications

“Python is used by Wikipedia, Google, Yahoo!, CERN and NASA, among many other organisations. YouTube, Instagram and Quora are among the countless sites that use Python. Much of Dropbox’s code is Python, Python has been used extensively by digital special effects house ILM (whose work spans across all of the Star Wars and Marvel films) and it’s a favourite of electronics titan Philips.” Some advantages of python are that it comes with extensive libraries,



# Lessons: Code.org

## Summary of Code.org Course

Summaries and descriptions below are primarily sourced from code.org

After completing this **Express course**, students should have the tools to be able to do the following:

1. Understand sequencing, loops, conditionals, functions, variables, and for loops
2. Understand when it is necessary and efficient to apply different coding concepts
3. Create solutions to programming tasks and challenges

After completing this **App Lab** course, students should have the tools to be able to do the following:

1. Understand the programming concepts of variables, conditionals, functions, loops, lists, traversals (for loops that iterate through lists), algorithms, functions with parameters and returns, inputs/outputs, and libraries (collections of functions)
2. Know best practices when it comes to applying the above concepts
3. Understand how to import data via tables into their programs instead of having to generate data themselves
4. Create a fully functional app with a Graphical User Interface (GUI) using the code, design, and data tabs and be able to share their app

After completing this **Web Lab** course, students should have the tools to be able to do the following:

1. Learn how to create and share the content on your own web pages
2. Learn how to structure and style your pages using HTML and CSS
3. Practice valuable programming skills such as debugging and commenting.
4. Have a personal website that you can publish to the Internet.

## Course 1: Express Course

[https://studio.code.org/s/express-2020?section\\_id=3243389](https://studio.code.org/s/express-2020?section_id=3243389)

### Sequencing

Lesson 2: Programming with Angry Birds

Lesson 3: Debugging with Scrat

Lesson 5: Creating Art with Code

### Loops

Lesson 6: Loops with Rey and BB-8

Lesson 7: Nest Loops in Maze

Lesson 9: Snowflakes with Anna and Elsa

Lesson 10: Looking Ahead with Minecraft

## **Conditionals**

Lesson 11: If/Else with Bee

Lesson 12: While Loops with the Farmer

Lesson 13: Conditionals in Minecraft: Voltage Aquatic

Lesson 14: Until Loops in Maze

Lesson 15: Harvesting with Conditionals

## **Functions**

Lesson 16: Functions in Minecraft

Lesson 17: Functions with Harvester

Lesson 18: Functions with Artist

## **Variables**

Lesson 19: Variables with Artist

Lesson 20: Changing Variables with Bee

Lesson 21: Changing Variables with Artist

## **For Loops**

Lesson 22: For Loops with Bee

Lesson 23: For Loops with Artist

Course 2: Javascript-based App Lab: CSP Unit 3-7, 9

<https://code.org/educate/applab>

## **Unit 3: Intro to App Lab**

Students design their first app while learning both fundamental programming concepts and collaborative software development processes. Students work with partners to develop a simple app that teaches classmates about a topic of personal interest. Throughout the unit, they learn how to use Code.org's programming environment, App Lab, to design user interfaces and write simple event-driven programs. Along the way, students learn practices like debugging, pair programming, and collecting and responding to feedback, which they will be able to use throughout the course as they build ever more complex projects. The unit concludes with students sharing the apps they develop with their classmates

## **Unit 4: Variables, Conditionals, and Functions**

Students expand the types of apps they can create as they learn how to store information (variables), make decisions (conditionals), and better organize code (functions). Each programming topic is covered in a specific sequence of lessons that ask students to 'Explore' ideas through hands-on activities, 'Investigate' these ideas through guided code reading, 'Practice' with sample problems, and apply their understanding as they 'Make' a one-day scoped project. The entire unit concludes with a three-day open-ended project in which students must build an app that makes a recommendation about any topic they wish.

### **Unit 5: Lists, Loops, and Traversals**

Students learn to build apps that use and process lists of information. Like the previous unit, students learn the core concepts of lists, loops, and traversals through a series of EIPM lesson sequences. Later in the unit, students are introduced to tools that allow them to import tables of real-world data to help further power the types of apps they can make. At the conclusion of the unit, students complete a week-long project in which they must design an app around a goal of their choosing that uses one of these data sets.

### **Unit 6: Algorithms**

Students learn to design and analyze algorithms to understand how they work and why some algorithms are considered more efficient than others. This short unit is entirely unplugged, and features hands-on activities that help students get an intuitive sense of how quickly different algorithms run and the pros and cons of different algorithms. Later in the unit, students explore concepts like undecidable problems and parallel and distributed computing.

### **Unit 7: Parameters, Returns, and Libraries**

Students learn how to design clean and reusable code that can be shared with a single classmate or the entire world. In the beginning of the unit, students are introduced to the concepts of parameters and return, which allow for students to design functions that implement an algorithm. In the second half of the unit, students learn how to design libraries of functions that can be packaged up and shared with others. The unit concludes with students designing their own small library of functions that can be used by a classmate.

### **Unit 9: Data**

Students explore and visualize datasets from a wide variety of topics as they hunt for patterns and try to learn more about the world around them from the data. Once again, students work with datasets in App Lab, but are now asked to make use of a data visualizer tool that assists students in finding data patterns. They learn how different types of visualizations can be used to better understand the patterns contained in datasets and how to use visualizations when investigating hypotheses. At the conclusion of the unit, students learn about the impacts of data analysis on the world around them and complete a final project in which they must uncover and present a data investigation they've completed independently.

Course 3: Web Lab: Web Development

<https://code.org/educate/weblab>

In the Web Development unit, students are empowered to create and share the content on their own web pages. They begin by thinking about the role of the web, and how it can be used as a medium for creative expression. As students develop their pages and begin to see themselves as programmers, they are encouraged to think critically about the impact of sharing information online and how to be more critical content consumers. They are also introduced to problem solving as it relates to programming, as they learn valuable skills such as debugging,



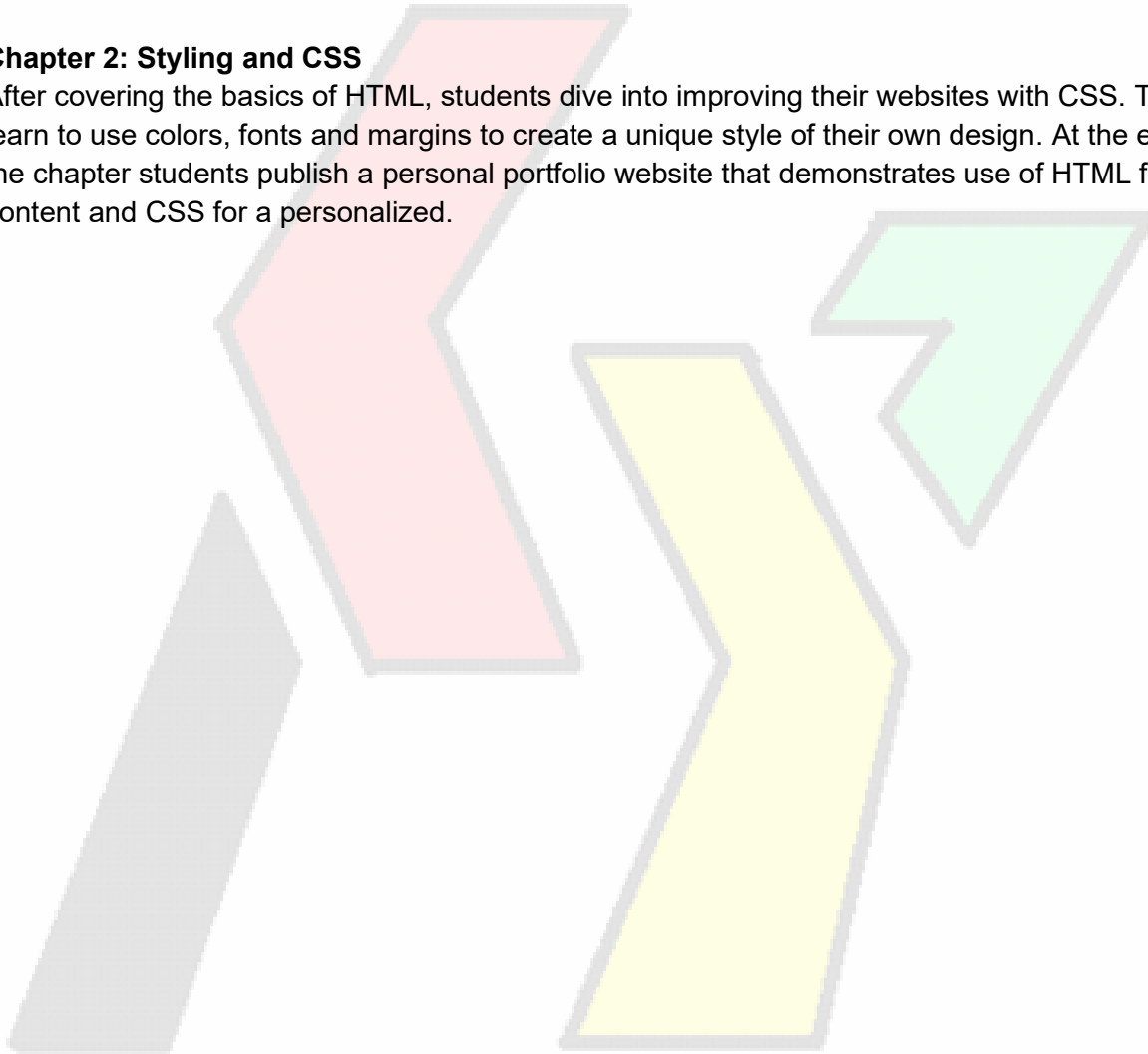
commenting, and structure of language. At the conclusion of the unit, students compile their work to create a personal website they can publish and share.

### **Chapter 1: Web Content and HTML**

Students use computing as a form of self expression as they design and develop basic web pages. Focusing on the tags, keywords, and syntax used to communicate instructions to the computer, students use HTML to structure the content of a web page. They also explore the privacy and intellectual property implications of publishing their work online.

### **Chapter 2: Styling and CSS**

After covering the basics of HTML, students dive into improving their websites with CSS. They learn to use colors, fonts and margins to create a unique style of their own design. At the end of the chapter students publish a personal portfolio website that demonstrates use of HTML for content and CSS for a personalized.



# Lessons: Python

## Summary of Python Course

After completing this **Python Course**, students should have the tools to be able to do the following:

1. Understand the programming concepts of sequencing, loops, conditionals, functions, classes, parameters, returns, variables, arrays (lists, dictionaries, and tuples), algorithms, inputs/outputs, for loops, and libraries (collections of functions and attributes)
2. Understand when it is necessary and efficient to apply different coding concepts
3. Understand valuable programming skills such as pseudocode, debugging and commenting.
4. Create solutions to programming tasks and challenges
5. Create solutions to real world problems that can utilize software solutions
6. Think logically and rationally by being able to deconstruct problems to their elemental components, address the needs of those components, and finally reconstruct a solution that addresses each of those needs
7. Address efficiency and feasibility concerns in constructing solutions due to the natural and artificial limits answers to programming inquiries can place on potential solutions.

### Lesson 1: Express Course -> Sequencing, Loops, and Conditionals

[https://studio.code.org/s/express-2020?section\\_id=3243389](https://studio.code.org/s/express-2020?section_id=3243389)

#### Sequencing

Lesson 2: Programming with Angry Birds

Lesson 3: Debugging with Scrat

Lesson 5: Creating Art with Code

#### Loops

Lesson 6: Loops with Rey and BB-8

Lesson 7: Nest Loops in Maze

Lesson 9: Snowflakes with Anna and Elsa

Lesson 10: Looking Ahead with Minecraft

#### Conditionals

Lesson 11: If/Else with Bee

Lesson 12: While Loops with the Farmer

Lesson 13: Conditionals in Minecraft: Voltage Aquatic

Lesson 14: Until Loops in Maze

Lesson 15: Harvesting with Conditionals

## Lesson 2: Introduction to Integrated Development Environments (IDE's), Commands, Data Types, and Variables

### IDEs & Resources

I would encourage you to show them both IDE's so that students who are experiencing difficulty keeping up can revert to Edublocks so that they do not fall behind. Depending on the level of user (age/experience), you can choose between the following IDEs:

- Colab -> <https://colab.research.google.com/>

### What is Colaboratory?

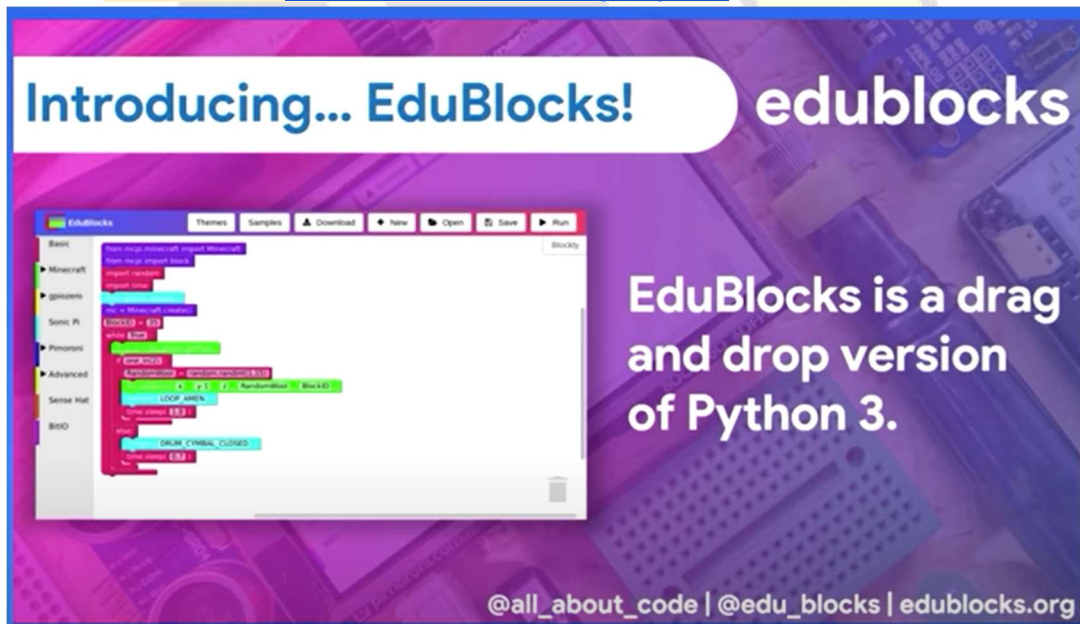
Colaboratory, or "Colab" for short, allows you to write and execute Python in your browser, with

- Zero configuration required
- Free access to GPUs
- Easy sharing

Whether you're a **student**, a **data scientist** or an **AI researcher**, Colab can make your work easier. Watch [Introduction to Colab](#) to learn more, or just get started below!

- This is a **text-based** IDE for python among other things and can even be used for machine learning programs.
- Intro to Colab Video -> <https://www.youtube.com/watch?v=inN8seMm7UJ>
- When you create your own Colab notebooks, they are stored in your Google Drive account.
- You can easily share your Colab notebooks with co-workers or friends, allowing them to comment on your notebooks or even edit them

- EduBlocks -> <https://app.edublocks.org/#Python>



The image is a promotional graphic for EduBlocks. It features a blue and purple background with a grid pattern. On the left, there's a screenshot of the EduBlocks interface showing a Python code editor with a sidebar of categories like 'Basic', 'Minecraft', 'Sensors', 'Sonic Pi', 'Periscope', 'Advanced', 'Sense Hat', and 'BBC'. The main text on the right reads 'Introducing... EduBlocks!' and 'edublocks'. Below that, it says 'EduBlocks is a drag and drop version of Python 3.' At the bottom right, there are social media handles '@all\_about\_code | @edu\_blocks | edublocks.org'.

- This is a drag and drop **block-based** version of Python 3.

- EduBlocks allows users to go back and forth between the blocks and text versions by clicking on the “blocks” or “python” button located on the upper right side of the screen
- The most important part about edublocks is that the actual python code is written on the blocks for the user to see including the indentations.
- This, just like the express course, will allow kids to focus on learning the foundational concepts like loops and conditionals for python while being exposed to the syntax without the frustrations of having to actually perform the syntax themselves

Finally, introduce your students to resources like [w3schools](#) which they can use for self-directed and self-paced development with python and other programming languages. Please notice all the other resources cited and linked throughout this document.

### Commands, Data Types, and Variables

They may get annoyed but you should have students repeat the challenges from this section at least 4 or 5 times so that they can build some muscle memory when it comes to the mindset and logic involved with these kinds of problems. It is important that they form a strong foundation with the basics

- Introduce commands and strings with the **print()** command
  - Print a **string**
  - Show **concatenation** of strings: Use adding string numbers vs **integer** numbers to demonstrate “sandwiching” vs mathematical addition
  - Show saving strings in **variables**
  - Show why saving strings in variables matter by printing out the variable a bunch of times because when you want to change the print **output**, rather than having to go and replace a string everywhere it occurs, you can just change it at the variable and the program will automatically propagate the changes
  - Ask students how you would add spaces between concatenated strings
  - Try to concatenate a string and an integer. Talk about the error that occurs. Show them the comma method instead
  - Show them casting data types using **int()** and **str()** so that they can concatenate
- Have them test all they have learned here with the following exercises
  - **Multivariable Print:** Have a variable pop up many times in a print statement and change it to see the results of the print statement change
    - Save a string in a variable
    - make up sentences where that variable needs to appear in multiple places
    - Print the sentences then change the variable
    - Print the sentences again after the change to see how the program responds
  - **Madlibs:** Create 5 sentences with at least 5 blanks. You will replace these blanks with variables:
    - 3 of the blanks have to be strings

- 2 of the blanks have to be integers
- You must first print out the “blank” version of the sentences
- You must then print out the “filled” version of the sentences with the blanks being replaced by your variables

### Lesson 3: User inputs, casting, basic logic, and pseudocode

- Introduce **input()** and reinforce casting
  - Show them how the input command works by requesting a user’s input and remind them that for them to use the input later on, it is best to save it inside a variable
  - Remind them that the input is saved as a string
  - Remind them that if they need to get a number from the input and they need to do math with the number, they have to cast it using int()
- Talk about **pseudocode**
  - According to [The Economic Times](#), pseudocode is an informal way of programming description that does not require any strict programming language syntax or underlying technology considerations.
  - It is used for creating an outline or a rough draft of a program.
  - Pseudocode summarizes a program’s flow, but excludes underlying details
  - System designers write pseudocode to ensure that programmers understand a software project's requirements and align code accordingly.
  - It uses short terms or simple English language syntaxes to write code for programs before it is actually converted into a specific programming language.
  - Advantages:
    - Anyone, including non-programmers, can read and understand pseudocode because it simply describes the logic of a program
    - It enables the programmer to focus on just the algorithm/reasoning part of their program and not the programming language or syntax itself
  - [Pseudocode example from Free Code Camp](#)
- Have them test all they have learned here with the following exercises
  - **Madlibs (again):** Create 5 sentences with at least 5 blanks. You will replace these blanks with variables:
    - 3 of the blanks have to be strings
    - 2 of the blanks have to be integers
    - You must first print out the “blank” version of the sentences
    - You must then print out the “filled” version of the sentences with the blanks being replaced by your variables.
    - The variables must be filled using input() instead of hardcoding.
    - Don’t forget you need to do something special for the number inputs
  - **Say Hi:** Ask someone for their name and say “Hi,” name, “nice to meet you”
  - **Birth year guesser:** Ask someone their age and what year it is and tell them what year they were born
  - **100th birthday guesser:** Ask someone how old they are and what year it is and

tell them when they will turn 100.

- **Knock Knock Jokes:** Look up knock knock jokes and create a program that tells them as the user types in the responses

## Lesson 4: Conditionals and Comparators

- Introduce comparators: they compare what is on the left to what is on the right and result in a True or False response also known as a Boolean data type in python
  - `a == b`: is "a" equal to "b"?
  - `a != b`: is "a" not equal to "b"?
  - `a > b`: is "a" greater than "b"?
  - `a >= b`: is "a" greater than or equal to "b"?
  - `a < b`: is "a" less than "b"?
  - `a <= b`: is "a" less than or equal to "b"?
- *Note you can use number data types with all of the comparators, but you can only use the equal and not equal with strings*
- Have them run some example print statements like the following just to see the results:
  - `print(5 > 10)`
  - `print(5 < 10)`
  - `print(5 = 10)`
  - `print(5 != 10)`
  - `print(5 == 10)`
  - `print(5 >= 10)`
  - `print(5 <= 10)`
- Introduce conditional (if-elif-else) statements and explain what role comparators play with conditional statements
  - Conditional statements are simple. If the condition (boolean condition of the if statement) is satisfied (condition = True), the code block underneath that section will run.
  - If the condition is not satisfied (condition = False), the code block underneath that section will NOT run.
  - If there are more conditions following that false condition (elif statement(s)), those will be checked as well and the first one to be satisfied (condition = True) will run the code block underneath it. Otherwise, the code block underneath it will NOT run because the condition is not satisfied (condition = False).
  - If none of the conditions are satisfied (False), the False condition block (else statement) will run. If there is no false condition block, then nothing will happen if none of the conditions of the if and elif statements are NOT satisfied (condition = False). The code will simply move on to whatever comes after the if/if-elif block.
  
  - If True: *# if false, move on to the next condition which is the "elif 1"*
  - it will run whatever code is here (if)
  - Elif 1 True: *# previous condition false so this is now being evaluated, move on to the next condition which is the "elif 2" if this (elif 1) is false*

- it will run whatever code is here (elif 1)
- Elif 2 True: # previous condition false so this is now being evaluated, move on to the next condition which is the “elif 3” if this (elif 2) is false
- it will run whatever code is here (elif 2)
- Elif 3 True: # previous condition false so this is now being evaluated, move on to the next condition which is the “else” if this (elif 3) is false
- it will run whatever code is here (elif 3)
- Else: # all conditions (if, elif 1, elif 2, elif 3) are False; therefore, run this block
- All conditions in this if block are False will run this code
- Have them run some example of if-elif-else statements to see how it works:
  - 3 simple math comparators
    - If `int(a) > int(b)`, print out that a is greater than b
    - If `int(a) > int(b)`, print out that a is greater than b, else print b is greater
    - If `int(a) > int(b)`, print out that a is greater than b, elif `int(a) < int(b)`, print b is greater, else print they are equal
  - Create a password login system that will tell you whether you entered the correct password or not
  - Create a password login system that will tell you whether you entered the correct password, tell you if you entered another password that was close enough, and then tell you if you got it completely wrong

## Lesson 5: Boolean/Logical Operators

- Introduce Boolean operators and explain: resources are primarily from [Real Python](#)

Python has three Boolean operators that are typed out as plain English words:

1. and
2. or
3. not

These operators connect Boolean expressions (and objects) to create compound Boolean expressions.

The Python Boolean operators always take two Boolean expressions or two objects or a combination of them, so they’re considered **binary operators**.

Operator	Description	Example	Try it
and	Returns True if both statements are true	<code>x &lt; 5 and x &lt; 10</code>	<a href="#">Try it »</a>
or	Returns True if one of the statements is true	<code>x &lt; 5 or x &lt; 4</code>	<a href="#">Try it »</a>
not	Reverse the result, returns False if the result is true	<code>not(x &lt; 5 and x &lt; 10)</code>	<a href="#">Try it »</a>

- **and**

- If both subexpressions evaluate to True, then the expression is True.
- If at least one of the subexpressions (exp1 or exp2) evaluates to False, then the expression is considered to be False

Result of exp1	Result of exp2	Result of exp1 <b>and</b> exp2
True	True	True
True	False	False
False	True	False
False	False	False

- **or:**

- If at least one of the subexpressions (exp1 or exp2) evaluates to True, then the expression is considered to be True.
- If both subexpressions evaluate to False, then the expression is False.
- This definition is called inclusive or, since it allows both possibilities as well as either.

Result of exp1	Result of exp2	Result of exp1 or exp2
True	True	True
True	False	True
False	True	True
False	False	False

**Table 1.** Logical Python or Operator: Truth Table



```
Python >>>
>>> exp1 = 1 == 2
>>> exp1
False
>>> exp2 = 7 > 3
>>> exp2
True
>>> exp1 or exp2 # Return True, because exp2 is True
True
>>> exp2 or exp1 # Also returns True
True
>>> exp3 = 3 < 1
>>> exp1 or exp3 # Return False, because both are False
False
```

In the previous examples, whenever a subexpression is evaluated to `True`, the global result is `True`. On the other hand, if both subexpressions are evaluated to `False`, then the global result is also `False`.

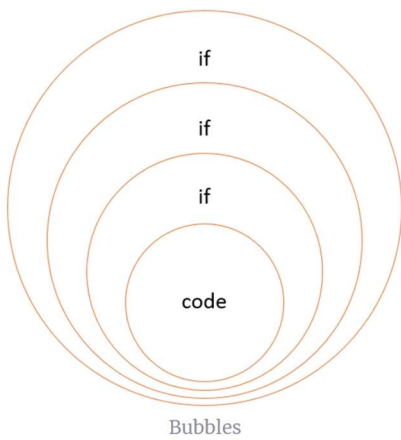
Images from [Real Python](#)

- **not**
  - Returns the opposite boolean value of the expression. If you have a boolean expression return `True` with **not** attached to it, the final return will be **False** and vice versa
- Have them run some examples to see how it works:
  - Come up with a question that asks the user to input a yes or no answer.
  - Create your program such that it allows for variations of yes or no like yea or nah

Add more logic exercises

### Lesson 6: Nested Conditionals

- Introduce Nested if-statements: An if-statement inside an if-statement
- According to [ExcelJet](#), Nested IFs are used when you need to test more than one condition and return different results depending on those tests.
- According to [WPSHout](#), deeply nested conditionals make it just about impossible to tell what code will run, or when; therefore, it is best to avoid these when possible and instead opt for smoother, more straightforward logic



**code**



- This is about not making the conditional checks dependent on each other if they do not have to be

```

if expression1:
    statement(s)
    if expression2:
        statement(s)
    elif expression3:
        statement(s)
    elif expression4:
        statement(s)
    else:
        statement(s)
else:
    statement(s)

```

```

#!/usr/bin/python
var = 100
if var < 200:
    print "Expression value is less than 200"
    if var == 150:
        print "Which is 150"
    elif var == 100:
        print "Which is 100"
    elif var == 50:
        print "Which is 50"
    elif var < 50:
        print "Expression value is less than 50"
else:
    print "Could not find true expression"
print "Good bye!"

```

When the above code is executed, it produces following result -

```

Expression value is less than 200
Which is 100
Good bye!

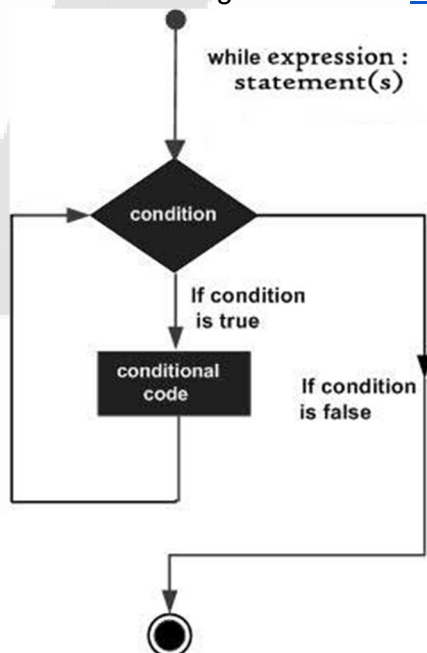
```

## [Images from Tutorials Point](#)

- Have them run some example of nested if statements to see how it works:
  - Create a multilayered password system that lets you guess the 2nd password only after you guess the 1st password correctly
  - Create a smart thermometer program that will first ask whether you're home and if you are, asks you what temperature you want to set. If you're away, it should have the temperature set at some default temperature
  - Create a number guessing game that lets you know if your wrong guess is too high or too low, gives you another chance to guess, and then evaluates your second guess, and then repeats the process for a 3rd and final guess.

## Lesson 7: While Loops

- Introduce while loops and why they are useful in creating efficient code
- The while loop syntax is similar to if statements in python
- It allows you to repeat your code over and over again until you achieve a desired outcome. That repetition is known as **iteration** in coding/programming and every instance of a while loop's code running is known as another iteration
- Structure:
  - if the while condition is satisfied (condition is True), the code inside the while loop will run.
  - After the code runs, it will go back up to the while loop condition and check it again to see if the while condition is satisfied (condition is True).
  - If the condition is true, the code inside the while loop will run again and the program will go back up to the while loop condition and check it again.
  - This process will repeat until the condition is False.
  - Once this happens, the program will skip to the code after the while loop.
  - This image below from [Tutorial's Point](#) demonstrates this process:



- Changing the while loop condition can be done in many ways: a user input, an updating variable like a counter as in the example below, etc.
- Proper while loop example from [W3Schools](#): In it, the program is counting up to 6 while updating the while loop condition based on the variable i

## Example

Print i as long as i is less than 6:

```
i = 1
while i < 6:
    print(i)
    i += 1
```

Try it Yourself »

- Infinite while loops:
  - If a satisfied while condition (condition is True) is not updated within the while loop, the condition will always be satisfied (condition is always True) and the while loop will run forever
  - Here's an example from [Tutorial's Point](#):

```
#!/usr/bin/python

var = 1
while var == 1 : # This constructs an infinite loop
    num = raw_input("Enter a number :")
    print "You entered: ", num

print "Good bye!"
```

When the above code is executed, it produces the following result –

```
Enter a number :20
You entered:  20
Enter a number :29
You entered:  29
Enter a number :3
You entered:  3
Enter a number between :Traceback (most recent call last):
  File "test.py", line 5, in <module>
    num = raw_input("Enter a number :")
KeyboardInterrupt
```

Above example goes in an infinite loop and you need to use CTRL+C to exit the program.

- Loop Control Statements: these cause the loops to react differently than a normal while

loop depending on how you use them

- Break: exits the while loop completely
- Continue: skips the rest of the executable code in the while loop on the current iteration and goes back up to evaluate the condition for the next iteration
- Pass: this does nothing. while loops need code in them to be syntactically sound in python and so “pass” is just code you can put in your while loop so that it has correct from.

```
→ while <expr>:  
    <statement>  
    <statement>  
    break  
    <statement>  
    <statement>  
    continue  
    <statement>  
    <statement>  
  
<statement>
```

- Nested loops: Programmers can nest loops in the same way they can nest conditional statements
- Have them do these exercises to practice while loops:
  - Create a while loop that counts down from 100 by 2, by 10, by 20
  - Create a while loop that counts up to 100 by 2, by 10, by 20
  - Create a password system that gives you an unlimited amount of chances to guess the password
  - Create a password system that gives you 3 chances to guess the password. You will need to use a break statement in here somewhere (you can decide to let students know about the break statement beforehand or have them do it without warning them and then when their code is running one extra time because of the missing break statement, point out that their program asks for the user input one extra time without evaluating it and ask them to address it. If they struggle with addressing that, then point them in the direction of the break statement)
  - Recreate the number guessing game from the conditionals lesson that tells you if your guess is too high or too low. This time, with an infinite amount of guesses

## Lesson 8: Libraries and Arrays/Lists

- Introduce libraries as a collection of functions that you can import into your program
- Demonstrate with [random library](#) in python
- Introduce arrays as a concept and how they let you store multiple items in a variable and their application in lists in python
- Talk about list indices and how to access items in a list, add items to a list, remove items from a list, etc.

Python has a set of built-in methods that you can use on lists/arrays.

Method	Description
<code>append()</code>	Adds an element at the end of the list
<code>clear()</code>	Removes all the elements from the list
<code>copy()</code>	Returns a copy of the list
<code>count()</code>	Returns the number of elements with the specified value
<code>extend()</code>	Add the elements of a list (or any iterable), to the end of the current list
<code>index()</code>	Returns the index of the first element with the specified value
<code>insert()</code>	Adds an element at the specified position
<code>pop()</code>	Removes the element at the specified position
<code>remove()</code>	Removes the first item with the specified value
<code>reverse()</code>	Reverses the order of the list
<code>sort()</code>	Sorts the list

- Have them do these exercises to practice the concepts of libraries and lists
  - Create a program that generates and prints random numbers
  - Create a program that generates and prints random strings
  - **More random functions**

## Lesson 9: For Loops

- Introduce for loops as a tool in python that iterates over a sequence (that is either a list, a tuple, a dictionary, a set, or a string) [w3schools](https://www.w3schools.com/python/python_for_loops.asp).
- With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.

```
# prints each fruit in the list
fruits = ["apple", "banana", "cherry"]
for x in fruits:
    print(x)
```

```
apple
banana
cherry
```

Result Size: 669 x 653

### Looping over a list example

```
# Loop through the letters in the word "banana":
for x in "banana":
    print(x)
```

```
b
a
n
a
n
a
```

### Looping over a string example

```
Run »  
# Exit the loop when x is "banana":  
fruits = ["apple", "banana", "cherry"]  
for x in fruits:  
    print(x)  
    if x == "banana":  
        break
```

apple  
banana

```
Run »  
# Exit the loop when x is "banana", but this time the break comes before the  
print:  
  
fruits = ["apple", "banana", "cherry"]  
for x in fruits:  
    if x == "banana":  
        break  
    print(x)
```

apple

## 2 Break statement examples

```
Run »  
#With the continue statement we can stop the current iteration of the loop, and  
continue with the next:  
  
# do not print banana  
fruits = ["apple", "banana", "cherry"]  
for x in fruits:  
    if x == "banana":  
        continue  
    print(x)
```

apple  
cherry

## Continue statement examples

# The range() Function

To loop through a set of code a specified number of times, we can use the `range()` function,

The `range()` function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

### Example

Using the `range()` function:

```
for x in range(6):  
    print(x)
```

Try it Yourself »

```
for x in range(6):
    print(x)
```

```
0
1
2
3
4
5
```

Using range() to set your code to run a specific number of times

- Have them do these exercises to practice for loops
  - Print out items from a list
  - **More for loop exercises**

### Lesson 10: Putting it all Together with Rock Paper Scissors

- Create rock paper scissors game
  - Use .lower(), .upper(), and .replace() functions to try and help players out by removing extra spaces, caps, etc. that would make python reject their answers

### Lesson 11: Functions

- Introduce functions as a block of code that can give commands or return information.
- You have to define functions and note that functions can also take parameters which are items you pass into the function for it to process
- Explain returns: it's like functions are submitting information back to the program by replacing the function call with the return information

```
def my_function():
    print("Hello from a function")

my_function()
```

```
Hello from a function
```

Creating a calling a function

```
def my_function(fname):
    print(fname + " Refsnes")

my_function("Emil")
my_function("Tobias")
my_function("Linus")
```

```
Emil Refsnes
Tobias Refsnes
Linus Refsnes
```

Defining a function with a parameter and passing an argument into the function

- Recursion: functions can call themselves too but be careful not to get stuck in an infinite loop



- Do these as practice for functions
  - Create an addition function
  - Create an subtraction function
  - Create a multiplication function
  - Create a division function
  - Create a function that gives you the max of a list
  - Create a function that gives you the min of a list
  - More function practice: <https://www.w3resource.com/python-exercises/python-functions-exercises.php>

## Lesson 12: Classes

### Class

A class is basically an outline or guide for creating objects. it gives it features known as *states* and functions known as *methods* like a video game character or map

### Object

An object is an instance of a class. So if the class is the blueprint, the object is when the blueprint comes to life.

Once you create an object, you can assign values to its attributes, you can retrieve values from it, you can access the methods inside the class, etc.

### Constructor: `init`

The `init()` method is a function that belongs to a class. What is important regarding `init()` is that it is called automatically every time you instantiate a new object from a class.

### Constructor: `self`

The first parameter of a method is always `self`. `Self` is not a keyword, you can actually name that first parameter whatever you like however conventionally it is called `self` and that's what other programmers expect to read in your code.

So the first argument is `self`, which is a reference to the object, not the class, but the object. When an object is created from the class, `self` will reference that object. It provides a way to make other variables and objects available everywhere in a class. The `self` variable is automatically passed to each method that is called through an object, which is why it is listed first in every method definition. Variables attached to `self` are available everywhere in the class.

1. Use these lessons from [w3schools](https://www.w3schools.com) to play with creating and modifying classes
2. Explore the python turtle library and use that library with the turtle class and environment class to give your students a chance to play and familiarize themselves with libraries.

- a. Here is an example of a turtle tutorial:

[https://www.youtube.com/watch?v=ogsRn1XSy5c&ab\\_channel=RazaZaidi](https://www.youtube.com/watch?v=ogsRn1XSy5c&ab_channel=RazaZaidi)



# Beyond the Lessons

## Future Development

At this point, they have learned a lot of what they can do in block, hybrid, and text based IDEs. Depending on how you structured your curriculum according to the framework, they may have experience with multiple languages. The rest is just about practice. This practice will bring up the need for them to discover specific skills making them relevant at that time like learning other languages, APIs, scraping, etc. This practice will also now begin to grow the ability for them to learn about new programming skills independently just like when they attempt to build their own software. This is important because 1) you cannot teach them everything there is to know about programming and 2) there will come a time where the world around programming will evolve and it is important for students to be able to keep up with or without a teacher present.

A good assignment for this is to tell them to come up with a program they may be thinking about developing and work to create it or work to recreate an app that already exists so you can reference it as you build your version. **This assignment will test both their problem solving and programming skills as they might not just copying a design, but they might be modifying it for the better, integrating advances from other creators, and beyond**

Ask your students to check out different programmers on youtube, Github, and other online platforms and see if there are any projects or areas they find interesting: cybersecurity, help desk, app development, software development, web development, etc. You can also allow students to bring you examples of things they would like to create and approve it based on the difficulty level. I rarely deny students a request; however, if I think it's too easy, I just ask them to pair it with another design so that they can get adequate practice

Students can also look into websites like [codewars](#) that give you a number of programming challenges with ranging difficulties in different programming languages to test your mind and help you achieve more proficiency with every challenge you overcome.

## Additional Tips for Instructor

1. This guide isn't the Bible. You can adjust it however you see fit to cater to your students' needs.
2. Don't gloss over reflections. They are probably the second most important part of this curriculum outside of practice time. Make time for them. Use class time to write AND discuss reflection questions. Come up with your own reflection questions that you feel are more relevant to your students if you like.
3. Give your students TIME to actually practice the skills you teach them. Don't try to rush things forward by giving them answers so that they can finish faster.

4. Don't save them from the struggle. Let them persevere and come to find the answers with you only being a guiding presence. Their search for answers will only deepen their understanding
5. You're not the reason they learn. You're just there to facilitate the process by creating an environment and challenges conducive to learning.
6. You will feel useless at times because the better your students get, the less they will need you. That is great. That means you are doing well because they are becoming more independent. It means you have successfully taught them how to learn

