



Software Development Learning Experience

Acknowledgements

This project is funded by a grant from the National Science Foundation to the University of North Carolina Greensboro (NSF Award # 1752436). Dr. Ryoko Yamaguchi is the principal investigator. Dr. Cyntrica Eaton is the primary author, workbook designer, and video producer. Adam Hall served as the editor.

Any opinions, findings, and conclusions or recommendations expressed in this report are those of the author and do not necessarily reflect the view of the National Science Foundation.



Software Development Learning Experience Overview

Preface: A Word to Parents, Caretakers, and Teachers.....	2
Preface: A Word to Students	3
Unit 1: The Top Three Reasons to Complete this Series	4
Unit 2: Mapping out the Road Ahead and Preparing for Growth (Part I).....	5
Unit 3: Algorithm Development for Beginners.....	8
Unit 4: Problem Solving and Planning for Algorithm Development	10
Unit 5: Program Implementation	14
Unit 6: Testing Your Code	18
Unit 7: Working with Turtle	19
Unit 8: Project.....	29
Unit 9: Next Steps	30
Unit 10: Reflecting on the Road Traveled and Measuring Growth (Part II).....	31
Appendix A: Terms to Know.....	33
Appendix B: Input-Processing-Output (IPO) Chart	34
Appendix C: The Python Interpreter.....	35
Appendix D: Graphing Support	36

To access the videos that accompany each unit, click on the corresponding icon:



Preface: A Word to Parents, Caretakers, and Teachers

Whether you serve as a parent, caretaker, or teacher to K-12 students, you have a front row seat as the young people in your life discover and attempt to make sense of their talents and interests. Helping young people channel their innate capabilities into careers that will grow and sustain them is one of your most important roles. However, for some career paths, it may not be clear how to steer students in the right direction and put them on track for success. This is especially true for careers in Computer Science (CS) where K-12 resources beyond [Scratch](#) are not widely accessible and exposure to critical pre-college CS learning experiences is limited. This series was designed to fill that gap as it pertains to an important role that (1) is critical to our technological future and (2) there is an endless need for in industry:

Software Developers.

We live in a digital world in which software systems are constantly being ideated, designed, and developed to satisfy our need to not only connect, collaborate, and complete tasks but to do these things quickly and more effectively via increasingly powerful systems. A diverse pipeline of talent – prepared to take on evolving technological challenges via forward-leaning innovation – must be nurtured to ensure that we maintain momentum. While an increase in adept software developers and computational thinkers will certainly help us remain competitive as a nation, the benefits aren't one-sided. Careers in software development are *lucrative* and provide an opportunity for students to *be creative, solve cutting-edge problems, and have global impact*. Given the projected talent shortage in the field, it is imperative that students of all walks of life are given full range to explore software development as a career and the opportunity to build the capacity to do so. Yet, this raises a key question given some of the challenges mentioned earlier: **how can we ensure that students with baseline interests and capabilities get *connected* to the resources needed to effectively prepare for a future as a software developer?**

This series was created to provide that *link*. More specifically, this series provides an opportunity for K-12 students to strengthen key foundational abilities, learn a programming language that is widely used by leading tech giants in industry, and prepare to take on advanced challenges in the future. These materials are geared toward middle school students – in part because they are at an inflection point in thinking about and planning for their futures and their careers. However, older students can also benefit from the learning exercises as well. The driving goal here is to provide a safe, informative space where students – regardless of race, gender, location, and socio-economic status – can develop a new muscle at their own pace and discover new ways to build on their interests while tapping into their natural capabilities. The units and activities that follow collectively make the process of learning how to code in Python more tangible and easy to navigate while simultaneously helping students develop skills, habits of mind, and the confidence necessary to go as far as their interests will take them.

We invite you to review the materials in this series, glance at what students will learn, and get a feel for how the materials are presented. As you do, think about young people who could benefit from this learning experience. Perhaps you know a student who wants to learn how to code or you want your student to get exposure to a wide range of careers including software engineering. Directing students to this series can help them discover their innate abilities and encourage them to set out on a path that spans from baseline interest to solid skills. Who knows, you might have the next tech genius in your midst! Let's work together to ensure that the next generation of developers and innovators have the materials they need to discover their capabilities and the foundation necessary to realize their full potential.

P.S. – Unit 1 centers around the [Top Three Reasons to Complete this Series](#). Please feel free to use this section to encourage students to participate!

Preface: A Word to Students

Imagine a career that allows you to be an innovator and a creative problem-solver working on new technological solutions; every day, you contribute to the development of game-changing systems. Imagine even further that this same career puts you in position to invent new technologies, launch tech-focused businesses, and impact your community (and even the world) in positive ways. If any of this sounds interesting, **software engineering** may be a career path worth exploring! But, interest in the field is only the first step. One of the most important next steps is discovering the answer to the following two-part question: **what skills do software developers use and how do they build them?** Getting the answers to these questions not only opens the door for in-depth exploration of this rewarding and lucrative career, it also helps you begin to understand how software development fits your own interests and aspirations. This series was designed to fill in the blanks by introducing fundamental concepts and providing a launch pad for you to develop and grow critical skills - all while teaching you **Python**, a language that is widely used in industry by leading tech giants.

One important point to make up front: because many of the software systems we use today can do *extraordinary* things, it's natural to think that the people who design these technologies are born with equally *extraordinary* super powers. Yet, in large part, the super powers that developers have – though seemingly magic – spring from very practical beginnings: interest in the field, access to and practice with the right tools, opportunities to strengthen computational thinking abilities, and the support necessary to grow your skills. This leads us to our next question – and it's one that fully captures the motivation for this series: what could **you** do if **you** had access to some of the same skills, tools, and support necessary for a career as a software engineer? **Let's find out together!**

To help you get the most out of this learning experience, the activities and videos that follow provide a safe space where you can develop skills **at your own pace** and discover new ways to build on your interests all while tapping into your natural capabilities. To get the full benefit, be sure to:

- **Use this workbook as a companion to your learning experience.** In addition to providing an introductory overview of topics, the workbook features exercises that deepen learning and allow you to practice applying newly acquired concepts and skills. It also helps you measure your progress and provides you with a tool to track how much you have learned over time.
- **Access and review the accompanying YouTube videos.** View the clips that accompany the units in this workbook (all of which are 7 minutes or less) to get an overview of key ideas covered, walk step-by-step through critical processes, and follow along with instructor-led examples. In addition to reinforcing the learning process, the videos allow you to watch, pause, and rewind as much as and whenever you need to. This helps to ensure that you can learn at your own pace and allows you to customize your learning experience.
- **Enlist a friend or a group of friends to go on this journey with you.** The process of learning to develop software is similar to a journey in a variety of ways (see Unit 2 for the full analogy). Having one or more friends to go on that journey with you, discuss the series with, bounce ideas off of, and compare notes with, can make the learning process more enjoyable and help to keep you on track. Ask a classmate or friend to go through the process with you. If you need help convincing them to join you, be sure to share the [Top Three Reasons to Complete this Series Out!](#)

We're excited that you've decided to take this journey. Remember, **turning the concepts** discussed in this series **into skill** takes **practice**. Using the resources provided here; diligently completing each unit – by yourself, with classmates, or with a friend; and working through the exercises will provide you with the experience, training, and support you need to build key skillsets and begin to explore whether software engineering is the career for you!

Unit 1: The Top Three Reasons to Complete this Series



Welcome to the **Software Development Learning Experience** – a carefully crafted suite of knowledge units, exercises, and videos designed to help students build, strengthen, and apply essential coding skills! If you're a 6th grader or higher, this series was especially designed to give you the space and support to develop key software development skills. In many cases, the missing link between you and a career as a software engineer is not *interest* or *raw capability* but **access**. Access to development tools. Access to training. Access to critical concepts and ideas. To fill that gap, this series provides access to each via a multi-media learning experience that will help you develop a strong knowledge base and – equally important – a foundational set of skills.

The **Software Development Learning Experience** is driven by the following principle: software development involves creativity and the imagination for sure, but at its base level, it integrates two critical skills – (1) problem solving and (2) language translation. In this series, we will use **computational thinking** to do our problem solving and the programming language **Python** in our translation efforts. Don't worry, you don't have to know about either approach or tool at this point. We'll take a deep dive into both later and add another critical skill to round out the conversation – stay tuned!

In all, the **Software Development Learning Experience** is a rich, empowering opportunity to build your knowledge base, your capabilities, and marketable skills all at the same time. But, to get the full benefit, you must complete each of the units that follow. Aside from the reasons already mentioned – here are three key reasons why you should!

1. **Learning new things makes you better at learning new things.** Much like our muscles get stronger with physical exercise, our brains get stronger with challenging cognitive exercises. In fact, our brains *thrive* on discovering new things, connecting dots, solving puzzles, and processing the world around us. As you learn and work through challenges, the brain expands its ability to process and store new information by creating new learning pathways and connections (called neurons) that facilitate deeper, more efficient knowledge acquisition and retention. This **Software Development Learning Experience** checks all the boxes by empowering you to discover new things while working through constructive challenges that require critical *and* creative thinking. Fully completing this full series will maximize these effects and the benefits will stay with you well into the future.
2. **The computational thinking approach applied in this series is relevant beyond Software Development.** Computational thinking is an approach to problem solving used to develop software but it also provides a way to think about, approach, and resolve problems outside of computer science. This series will not only teach you a key approach but also provide rich opportunities to use it.
3. **Software Development is an intriguing, lucrative skill and field.** The ability to write code will not only put you in the position to develop software solutions in the short term but to also have a lucrative career in the long term. In 2020, for example, the median salary for software developers was over \$110,000 a year. This series provides a toolkit you can use to grow critical skills, apply them, and make yourself competitive for this career in the future.

To sum it up, completing this series offers a host of short- and long-term benefits that can help you grow brainpower, learn new approaches to problem solving, and build lucrative skills. In the short term, you will grow your vocabulary, apply computational thinking, and develop computer programs using Python. In the long term, you will increase your capacity for learning, illuminate a career pathway, and put yourself in position to go as far as your interest will take you! But, we have to start somewhere. With that in mind, let's dive into the next unit!

Congratulations! You have finished [The Top Three Reasons to Complete this Series](#). Let's move on to [Mapping out the Road Ahead and Preparing for Growth \(Part I\)](#).

Unit 2: Mapping out the Road Ahead and Preparing for Growth (Part I)



The process of learning how to develop code is a lot like a journey. It involves navigating a (sometimes uncertain) path from a start point to a destination and – most importantly – it can also be quite transformational! In other words, with the right attitude, a little direction, and the right tools, you can *really* go places!

But much like the journey of a thousand miles begins with a single step, the journey of acquiring computer programming skills begins with small, incremental efforts that build over time and – with consistency – lead to significant results.

ProTip #1: Any time a journey feels intimidating or *too big*, try tackling it by shifting your focus from how long or hard the trek might appear to be to simply concentrating on the step directly in front of you. Focusing your attention on your *most immediate next step* allows you to have tangible success in the short term, build confidence in your abilities, and simultaneously keep moving forward toward the long term goal. **Win, win, win!**

The sole purpose of this unit is to provide you with a vision for the journey ahead. To help shape that vision, this unit provides a description of (1) the **pathway** we will take, (2) the **tools** we will use, and (3) a way to **track the growth** you will experience once the journey is complete. The last item is important because, as you navigate this learning experience, you will discover new concepts *in addition to* new things about how you learn, your abilities, and your interests. Capturing what you know and think **before** starting the journey and comparing it to what you have learned **after** completing the learning experience can be a powerful way to see how much your skill level and knowledge have evolved over time. We will start that process in this unit.

The Pathway

As we ease into the process of developing computer programs, we will focus on **three major steps**:

1. figuring out the **algorithm**, or the steps the computer will take to complete a task, **in English**,
2. translating that algorithm into a format, or **language**, the computer *understands*, and
3. testing the resulting computer program to ensure it produces the desired result.

These three steps, namely **algorithm development**, **program implementation**, and **software testing**, shape the pathway ahead and, in tandem, the units that follow. As we navigate the **Software Development Learning Experience** together, we will use **computational thinking** in step 1 to support algorithm development, **Python** in step 2 to translate the algorithm into a program, and the **Python interpreter** in step 3 to test the resulting program. Mastering these three aspects of software development and truly understanding how to accomplish each will provide a foundational platform that is strong enough to support future growth and stable enough for you to branch out as you explore and learn other languages and technologies.

To help solidify your vision of the journey ahead, the table below highlights how we will explore each step along with corresponding units.

Software Development Task	Learning Goal	Unit(s)
Algorithm Development	Understand and apply computational thinking for algorithm development	3 & 4
Program Implementation	Translate an algorithm from English into Python	5
Software Testing	Test the resulting Python program to ensure it behaves properly	6
Practice	Develop and test a drawing program using Python	7
	Create a program of your choice using the approaches and tools learned	8

Table 1: Overview of the steps covered in the Software Development Learning Experience

The Tools

Understanding how to develop software is as much about learning critical skills as it is about getting acquainted with a set of tools, concepts, vocabulary [words](#), and approaches. To support your development of key skills in algorithm development (computational thinking), program implementation (language translation), and software testing (quality control), we will use the following, each of which will be introduced in the units shown:

- [Input-Processing-Output \(IPO\) Chart](#) (Unit 3)
- [The Python interpreter](#) (Unit 5)
- [Graph paper](#) (Unit 7)

What these tools are and how they can be applied in software development will be discussed at length later. In the meantime, other resources that will help you along in this process include:

- This workbook
- The YouTube videos that accompany Units 1 - 10
- Pen and paper

Combined, the **workbook**, **YouTube clips**, and **ability to keep notes** will help to make the learning process easier and provide you with the support you need to not only build a strong foundation but nurture good habits as well.

As mentioned earlier, a list of [words](#) used in software development have also been included for your review. These [Terms to Know](#) were included in Appendix A because getting acclimated to a field is not only about building skills – building your vocabulary and understanding field-specific vernacular is equally important. With this in mind, be sure to review the terms provided and try to use as many as you can when talking to others about your experience. In addition, if you choose to explore software development beyond this learning experience, you will likely see these words used again and again. It's a good idea to get comfortable with them now.

The Transformation

One of the most fulfilling aspects of learning can be realizing how much your thinking and knowledge base have evolved over time. One way of doing this is by taking stock of where you started before a learning experience and comparing that with where you end up. To capture the **before** and get a snapshot of where you're starting, the next page of this workbook contains a list of 8 questions. **Answer the questions as best you can** keeping in mind that it is OK not to know all of the answers at this point. In fact, moving forward, you should consider that errors and mistakes are not failures; they indicate a knowledge gap and provide an opportunity to figure out what you need to either learn or review to improve your understanding of a topic. **Pro Tip #2**: Don't be afraid of mistakes or even failure. Making mistakes and even failing are rich opportunities to grow, enhance your capabilities, improve self-awareness, and launch a comeback.

Track Your Progress: Answer the following questions to the best of your current ability, based on what you know right now. You will answer the same questions at the end of Unit 10 to measure how much you have learned over time.

1. What are the three main processes used for developing computer programs?
2. What is an algorithm?
3. What is Python? What is it used for?
4. In computer programming: what is an interpreter and what is it used for?
5. What can you use to create graphics in Python?
6. What aspects of your personality or interests might serve you well as a software developer?
7. On a scale from 1 – 10, rate your ability to code in Python.
8. On a scale from 1 – 10, rate your general interest in programming.

Congratulations! You have finished [Mapping out the Road Ahead and Preparing for Growth \(Part I\)](#). Let's move on to [Algorithm Development for Beginners](#).

Unit 3: Algorithm Development for Beginners

Computers are amazingly powerful machines capable of storing and analyzing incredible amounts of data at speeds and magnitudes that far surpass the capabilities of the human brain. The way that we tap into and leverage that power is by providing computers with the step-by-step *directions* they need to perform key tasks. Those directions, called algorithms, are the basis of the code we will develop. Understanding how to develop algorithms is a critical part of software development. But, this raises the question: **how should we go about developing algorithms – especially if we’ve never done so before?** This unit addresses that question with an effective problem-solving tool that emphasizes computational thinking and makes algorithm development easier to manage.

Introduction to Algorithm Development

In general, problem-solving techniques help you reframe a problem, break it into smaller or more manageable parts, organize your thoughts, concentrate your efforts on one specific part of the problem, or all of the above. **ProTip #3:** Problem-solving techniques that help you get ideas out of your head and onto a sheet of paper have an added bonus. By allowing you to do a brain dump, the cognitive effort that would have been devoted to managing information in your mind can now be invested in deeper problem exploration. This can ultimately free up critical cognitive resources and, in turn, improve your ability to make sense of the problem, see the problem in a different view, and identify the kinds of connections that will make the solution set more obvious and easier to derive.

The problem-solving device we will use to develop algorithms is called an **Input-Processing-Output (IPO) chart** and to understand how it works, we will use one to solve the following problem:

Make a peanut butter and jelly sandwich (PB&J).

To see an *empty IPO* chart, jump to [Appendix B](#). A *completed IPO* chart for the peanut butter and jelly sandwich problem is shown below. Notice that it is comprised of three columns, each of which are labeled as either **Input**, **Processing**, or **Output**. In general, the columns keep track of the following information for a given problem:

- **Input:** the information, tools, devices, and resources that are necessary to implement the solution.
- **Processing:** an ordered list of instructions necessary to solve the problem.
- **Output:** the expected end-result(s).

With that in mind, the **IPO chart** that provides **the solution for making a PB&J** could be formatted as follows:

Input	Processing	Output
<ul style="list-style-type: none"> • Two slices of bread • One jar of peanut butter • One jar of grape jelly • A level surface • One knife • One napkin 	<ol style="list-style-type: none"> 1. Place the napkin on the level surface (i.e., table, countertop) and place the two slices of bread on top of the napkin, side-by-side. 2. Pick up the slice of bread on the left, dip the knife in the peanut butter, spread the peanut butter on the bread, and place the bread back on the napkin. 3. Wipe or rinse the knife. 4. Pick up the slice of bread on the right, dip the knife in the jelly, spread the jelly on the bread, and place the bread back on the napkin. 5. Place the left-most bread slice (covered in peanut butter) on top of the right-most bread slice (covered in jelly). 6. Slice the resulting sandwich in half. 	<ul style="list-style-type: none"> • A PB&J sliced in half

Table 2: IPO chart that documents how to assemble a peanut butter and jelly sandwich (PB&J)

It is important to note that although the **Processing** column of the IPO chart shown on the previous page is useful for this sample problem it is not *technically* complete. In particular, it is missing key steps that include listing how all of the input items are gathered. As an example, the jelly may have been retrieved from the refrigerator, however this step has not been accounted for in the **Processing** column. In addition, the **Processing** column for this example does not precisely indicate the amount of peanut butter or jelly used in the process. **The key take-away:** the IPO chart should be as precise as possible, however, as you complete the exercises that follow **in this unit**, use the example above to gauge how precise your solution should be. This will help to ensure that you practice using an IPO chart without getting lost in too many details or steps.

This unit is the first of two devoted to algorithm development; it was designed to ease you into the thought process of developing an IPO chart. In the next unit, we will explore the IPO chart more, focusing specifically on computer programs.

Unit 3 Exercises

1. Write a summary of what you learned in this unit in your own words.
2. What is an algorithm?
3. Develop an IPO chart for putting socks and tennis shoes with laces on each foot using the empty chart in [Appendix B](#).

Congratulations! You have finished [Algorithm Development for Beginners](#). Let's move on to [Problem Solving and Planning for Algorithm Development](#).



In the previous unit, we explored the Input-Processing-Output (**IPO**) **chart** as a problem-solving tool and applied it towards two real life activities: (1) making a peanut butter and jelly sandwich (**PB&J**) – together in the video – and (2) putting on shoes and socks – individually as an exercise. In this unit, we’re going to build on that knowledge with a specific focus on using the IPO chart to develop **algorithms** for computer programs.

To pivot towards software development, instead of using an IPO chart to document the process of making a PB&J, we will solve the following **sample problem**:

Construct a computer program that calculates the perimeter of a rectangle.

To begin developing the algorithm for this solution, we will conduct the following steps:

Planning and Algorithm Development Overview

1. Make sure you have a clear understanding of the problem you are trying to solve.
2. Draw a sketch of what the program will look like when it is running properly.
3. Carefully label each line in the screen with a letter; you will use these letters to develop your algorithm.
4. Construct your IPO chart.

In the remainder of this unit, we will execute steps 1 - 4 above in the context of the sample problem. As a recommendation: read through the remainder of the unit first, make a note of questions that arise or highlight anything that is unclear, and *then* take a look at the accompanying video clip that walks through the process step-by-step. Reading through the unit first, before reviewing the clip, will give you a quick introduction to the material. The questions that arise during the reading will likely be answered during review of the clip.

Planning and Algorithm Development Step-by-Step

1. **Make sure you have a clear understanding of the problem you are trying to solve.**

This step is *immensely important* and skipping it, especially when you’re starting to learn how to build algorithms, is counterproductive. As a developer, **your very first step in creating an algorithm should be solving the problem, by hand, on paper first.** Doing this properly ensures two interrelated, yet independently critical, things: (1) you will think through the process for solving the problem and (2) you will become deeply aware of the steps necessary for arriving at a solution. In some cases, to fully understand how to solve a problem, you may need to do some background research. For our sample problem, if you don’t know the equation for calculating the perimeter of a rectangle, looking it up should be one of your first steps in attacking the problem. Whether from research or from prior knowledge, before you begin, you will have to know that the proper equation for calculating the perimeter of a rectangle is:

$$\text{perimeter} = 2 * \text{length} + 2 * \text{width}$$

Having the equation in front of you and looking at it carefully provides some very important clues. Note how you need to have the values for both `length` and `width` before `perimeter` can be calculated. This indicates that the length and width are necessary **input** (or data needed to perform a certain task/calculation) and the perimeter is the **output** (or the expected end-result).

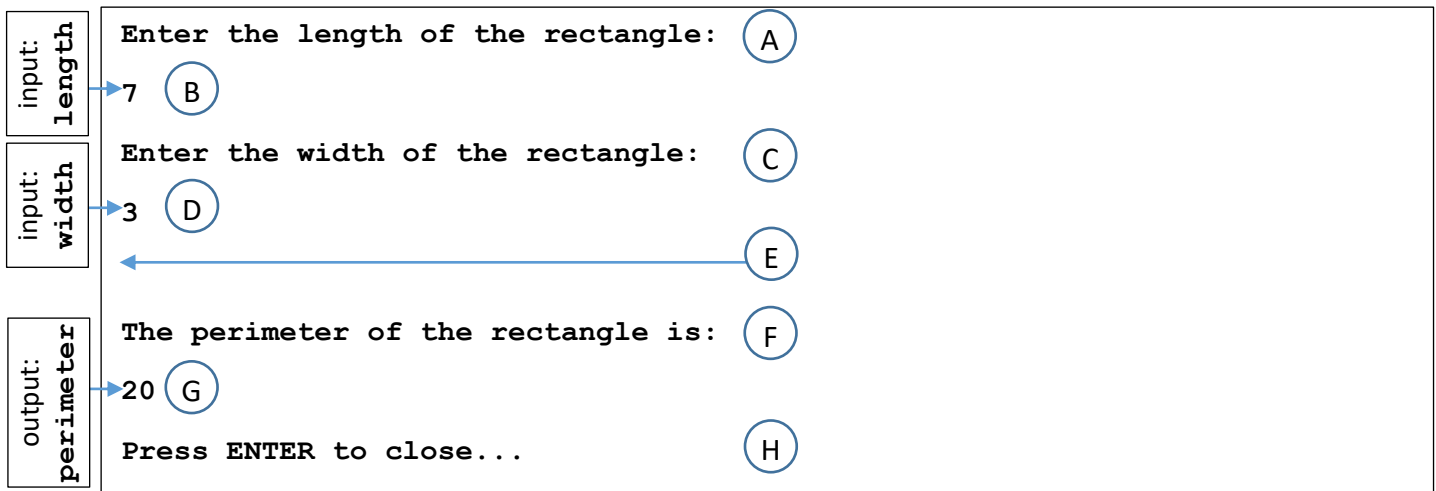
2. Draw a sketch of what the program will look like when it is running properly.

After working out the solution by hand, on paper in step 1, your next step is to grab another sheet of paper and draw what you expect the screen to look like when the program is complete. For this sample problem, a computer program that runs as follows would be ideal:

```
Enter the length of the rectangle:
7
Enter the width of the rectangle:
3
The perimeter of the rectangle is:
20
Press ENTER to close...
```


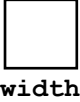

Being able to visualize what a Python program looks like in action is an *acquired skill* but the screen represented above and the step that preceded it are a blueprint that you can refer to in the future. More information about the sketch is included in the video that accompanies this chapter.

3. Next, label each individual line in your sketch with a letter and clearly label the input (length and width) and output (perimeter); later, each letter will align with a step in the Processing column of your IPO chart. Take a look at the labeled screen below and note the flow. Starting at the top, the program prompts (or asks) the user to enter a specific value (line A), gets the data and stores it in memory for later use as soon as the user taps the ENTER key (line B); these steps are then repeated, in the same order, in line C and line D. In line E, the program calculates the perimeter using the values collected in line B and line D. The program then presents the user with the results of the calculation by first printing a label (line F) and then the actual value (line G). The program closes by printing "Press ENTER to close..." on screen and waiting for the ENTER key to be pressed (line H). Again, it's important that you understand the flow of this process because it provides the basis for just about any program that calculates a value. In other words, once you understand the flow, you can modify it to solve similar problems in the future. Being able to observe examples like these, identify basic patterns, and apply those patterns to other problems are important skills in software development.



4. Next, use the labeled screen generated in step 3 as the basis for your IPO chart.

Note how the work completed in prior steps directly influences the steps that follow. The IPO chart below is directly linked to the sketch in step 3 (above). The values delineated as input show up in the **Input** column; the value delineated as an output shows up in the **Output** column; and an English description of what is happening in **lines A – H** shows up in the **Processing** column.

Input	Processing	Output
 	<p>A. Prompt the user to "Enter the length of the rectangle: "</p> <p>B. Wait for the input, convert it to an integer, and store it in length.</p> <p>C. Prompt the user to "Enter the width of the rectangle: "</p> <p>D. Wait for the input, convert it to an integer, and store it in width.</p> <p>E. Calculate the perimeter of the rectangle as $(2 * \text{width} + 2 * \text{length})$ and store the result in perimeter.</p> <p>F. Print "The perimeter of the rectangle is: "</p> <p>G. Print the value in the variable perimeter.</p> <p>H. Print "Press ENTER to close... " and wait for ENTER to be pressed.</p>	

At the end of this step, you have your algorithm (steps A – H) in English in the **Processing** column of the IPO chart and an idea about the number of inputs and outputs based on the **Input** and **Output** columns. The number of inputs and outputs is a key indicator of the number of **variables** you will need to hold user input and the results of any calculations. **Variables are computer memory locations designated to hold specific data.** We will talk more about them in the next unit.

An Important Word About Algorithm Development

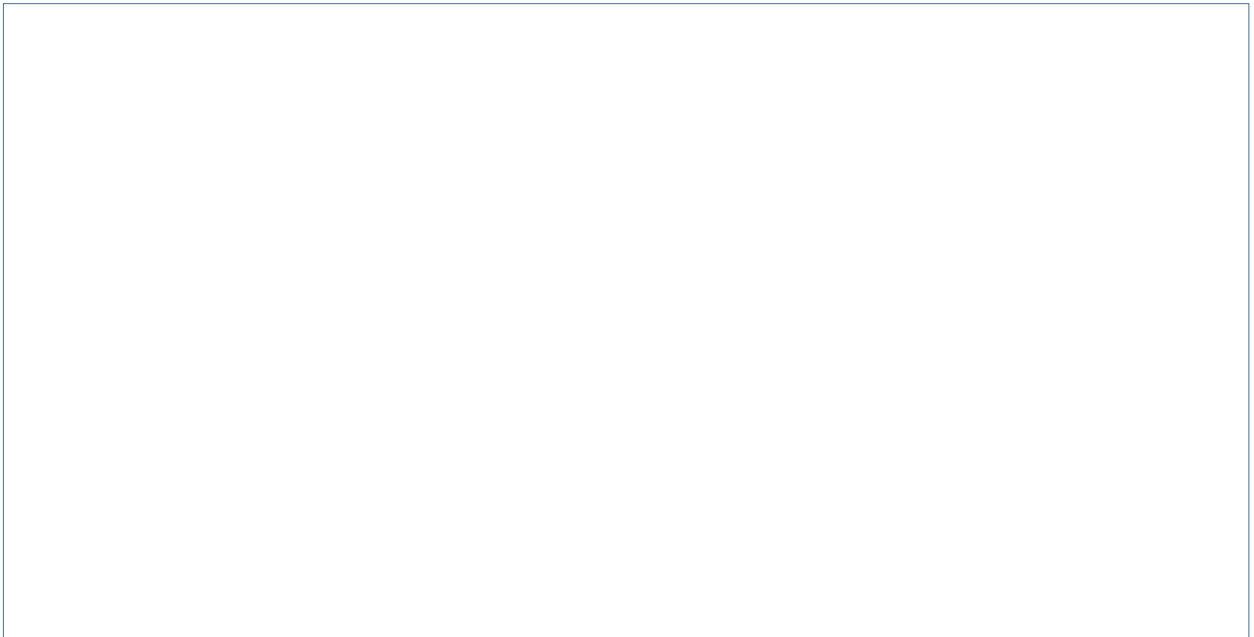
The process we've featured here can feel very time consuming when you're first starting out and, as a result, many budding developers opt to skip it and jump right into the highlight of the next unit – implementing the code. But, jumping directly into coding without a clear plan can lead to an inefficient process, shaped by trial and error, as opposed to precise problem solving. While trial and error has its merits, it has the potential to leave you frustrated and can take longer than other approaches to come to a solution. Sharpening your problem-solving skills by applying them during algorithm development can help to make the process more efficient and prepare you to handle more problem-solving tasks in the future.

Completing the algorithm development process outlined above and carefully learning how to implement each step provides you a *critical opportunity* to establish a *strong foundation* for software development. The analysis and planning inherent to this process will serve you well as an innovator developing forward-leaning software systems. **Over time, you will get faster at it.** **ProTip #4:** Any time you are learning a new concept, don't focus on speed. Instead, channel your time and energy into learning the process and allow yourself the space for critical ideas to sink in. With the ProTip in mind, work through the exercises on the next page to get more practice with algorithm development.

Unit 4 Exercises: We've covered a lot in this unit. To make sure that your knowledge of the algorithm development process sticks, it is absolutely imperative that you apply what you've learned within a day or two of completing the lesson (roughly 24 to 48 hours). Practicing the algorithm development steps with a new problem will challenge your brain and expand your learning capacity. These benefits will last well beyond completion of this learning experience.

1. Suppose you wanted to develop a computer program to calculate the area of a rectangle. What equation would you need?

2. Sketch a sample screen for a computer program that calculates the area of a rectangle.



3. Add labels to the screen sketch created in the previous step with for each line, input values, and output values.

4. Create an IPO chart based on the labeled screen created in the previous step; see [Appendix B](#).

Congratulations! You have finished [Problem Solving and Planning for Algorithm Development](#). Let's move on to [Program Implementation](#).

Unit 5: Program Implementation

In the previous unit, we worked through the algorithm development process for software development beginning with *solving the problem by hand* and ending with a *completed IPO chart*. As a result, at the end of Unit 4, we had an algorithm written in English in the **Processing** column of the IPO chart (steps A – H) and an inventory of the inputs and outputs needed based on the **Input** and **Output** columns. As you may remember, the Input and Output columns help us to understand how many memory locations, or variables, we will need to store critical information. This will make more sense in the section that follows.

Pro Tip #5: Completing algorithm development in advance of generating code is immensely important because once we have the IPO chart, we have the raw materials needed to develop our software. From there, the next step is to translate the IPO chart into a format or language the computer understands. This process is called **program implementation** and it is the focus of this unit.

Because we will be using **Python** as the formatting or programming language, a critical implementation tool for Unit 5 is the **Python interpreter**. To access the interpreter, you will need to download it. Visit [Appendix C](#) for directions.

This actually leads to a very important point. For the most part, the problem-solving techniques we used in the previous unit were language-independent. As a result, theoretically, we could use the work that we did in Unit 4 to create a program in any number of programming languages including Java, C++, or MATLAB. We will talk more about this later!

For now, our focus is to complete program implementation. We will do this using the following approach:

Implementation Overview

1. Start a [new python program](#) and begin implementation by adding comments in your code that document the author, the date, and the purpose of the program.
2. Type in the steps listed in the **Processing** column of the **IPO chart** as comments in your Python code.
3. Type in the Python code that actually implements each comment.

In the space that follows, we will implement each of the steps above in Python to solve our sample problem.

Implementation Step-by-Step

1. **Start a new Python program and begin implementation by adding comments in your code that document its author, the date, and purpose.**

In coding, comments are used to document important insight about code; since programming languages can be somewhat cryptic, comments provide human readers with informative notes about aspects of the code, including the algorithm. Ensuring that your code has thorough, well-formed comments is good programming practice. **Get used to integrating comments by always starting program implementation as follows:** create a blank file and then put in comments that document (1) who wrote the program, (2) the date the program was started, and (3) what the code is designed to do.

The code snippet on the next page provides the blueprint for what you should type into **Python's script mode window** (see [Appendix C](#)) when beginning implementation. **Pro Tip #6:** Be sure to type everything exactly as it is shown; any typos can result in interpreter errors – as we will discuss in Unit 6. Also, to properly customize the

code, replace [type your name] with your actual name and [type today's date] with today's actual date.

```
# Author: [type your name]
# Date: [type today's date]
# Purpose: Calculate the perimeter of a rectangle.
```

2. Type in the steps listed in the Processing column of the IPO chart as comments in your Python code.

In this stage, the goal is to extend the file created in the previous step by adding each line in the Processing column of the IPO chart as a comment. Note how there is direct alignment with lines A – H in the Processing column of the IPO chart we developed in Unit 4 and the newly added comments (shown below in blue).

This approach drives home the point that program implementation should follow algorithm development and directly leverage the IPO chart developed during that process.

```
# Author: [type your name]
# Date: [type today's date]
# Purpose: Calculate the perimeter of a rectangle.

# Prompt the user to "Enter the length of the rectangle: "
# Wait for the input, convert it to an integer, and store it in length.
# Prompt the user to "Enter the width of the rectangle: "
# Wait for the input, convert it to an integer, and store it in width.
# Calculate the perimeter of the rectangle as (2*width + 2*length)
# and store the result in perimeter.
# Print "The perimeter of the rectangle is: "
# Print the value of the variable perimeter.
# Print "Press ENTER to close... " and wait for a key press.
```

3. Type in the Python code that actually implements each comment.

Once the comments have been entered in properly, our next step is to type in the specific code that implements each comment. In effect, this is the step where we translate each of the English steps of the algorithm into a line of code that the computer understands. As shown below, the newly added code (in blue) translates the English represented in the IPO chart to Python.

```
# Author: [type your name]
# Date: [type today's date]
# Purpose: Calculate the perimeter of a rectangle.

# Prompt the user to "Enter the length of the rectangle: "
print("Enter the length of the rectangle: ")

# Wait for the input, convert it to an integer, and store it in length.
length = int(input())

# Prompt the user to "Enter the width of the rectangle: "
print("Enter the width of the rectangle: ")

# Wait for the input, convert it to an integer, and store it in width.
width = int(input())

# Calculate the perimeter of the rectangle as (2*width + 2*length)
# and store the result in perimeter
perimeter = 2 * width + 2 * length

# Print "The perimeter of the rectangle is: "
print("The perimeter of the rectangle is: ")

# Print the value of the variable perimeter.
print(perimeter)

# Print "Press ENTER to close..." and wait for ENTER to be pressed
input("Press ENTER to close... ")
```

Once step 3 is completed, we have officially planned and implemented our code! This is a significant milestone but, as mentioned earlier, we're not finished. Now, we need to make sure that our code runs properly. We'll learn how to do that in the next unit.

Exercise: This unit was primarily about translating process steps from English into Python. For each comment 1 - 5, determine what the Python code should be and write it underneath in the space provided.

1. # Prompt the user to "Enter your first name: "

2. # Calculate the sum as a + b

3. # Print "Today is a great day! "

4. # Print "The program is paused." and wait for the ENTER key to be pressed

5. # Print "Enter your name: ", wait for the input, and store it in name

Congratulations! You have finished [Program Implementation](#). Let's move on to [Testing Your Code](#).

Unit 6: Testing Your Code

Once you have completed program implementation, it is imperative that you test your code to ensure that it (1) compiles and (2) produces the expected results. To briefly elaborate, programs will not compile if there are **syntax errors** in them; these errors arise when a line of code has been typed in incorrectly. Consider the line below:

```
Print("Enter the length of the rectangle: ")
```

It has at least **two syntax errors**:

1. **The "P" in print is capitalized.** Python is a case-sensitive language, which means that it makes distinctions between uppercase and lowercase letters. As a result, the Python interpreter considers `print`, `Print`, and `PRINT` to be **three** different words even though they look very similar to the human eye. **The key take-home:** having an uppercase letter where a lowercase letter is expected causes interpreter errors and is a straightforward example of both Python's case-sensitivity and a syntax error.
2. **The final parenthesis is missing in the line.** This is a very common error and must be avoided. Make sure code components that need an open and a close, like parentheses and quotation marks, have both in place.

Pro Tip #7: Whenever code is not entered in the precise format that Python expects, it will be problematic. To avoid syntax errors, be careful to ensure your code has been formatted properly and address errors as they arise.

In addition to helping you catch and correct syntax errors, testing also allows you to ensure that your code does not have **logical errors**. These errors arise when a program is syntactically correct but something (like an ill-formed calculation) causes it to produce incorrect output. As an example of this, assume that, in our perimeter program, the user indicated the rectangle had a length of 10 and a width of 5. Given the equation, the perimeter would be:

```
perimeter = 2 * length + 2 * width
           = 2 * 10 + 2 * 5
           = 30
```

Consider the instance where your perimeter calculating program accepts 10 for the value of length and 5 for the value of width, but incorrectly calculates the perimeter as 19. This would be a solid indicator that you have a logical error somewhere in your program that needs to be corrected. This is another reason why you should always work a problem out on paper anytime you are trying to develop a computer program. It is important that you understand what the expected results should be to ensure that your program is running properly.

To detect if there are either syntax or logical errors in your programs, conduct the following steps:

Testing Overview

1. [Compile](#) the source code.
2. Correct any errors in the source code.
3. Repeat steps 1. and 2. above until the code compiles properly.
4. Make sure the program has the correct results (output); if the results (output) are incorrect, you may need to adjust/revisit your algorithm.

While this unit is comparatively shorter than the others, it is no less important. Ensuring that you have produced high-quality code that compiles and produces the correct result is paramount. Testing is the only way to do this.

Congratulations! You have finished [Testing Your Code](#). Let's move on to [Working with Turtle](#).

Unit 7: Working with Turtle

Up to this point, we have focused our attention on developing code for a program that calculates a value (i.e., the perimeter) based on user input. However, Python also has the capability to draw graphics. In this section, we will explore how to create graphic-oriented programs in the context of the development process we applied earlier.

As a reminder, here are the major steps of the development process:

Planning and Algorithm Development Overview

1. Make sure you have a clear understanding of the problem you are trying to solve.
2. Draw a sketch of what the program will look like when it is running properly.
3. Carefully label each line in the screen with a letter; you will use these letters to develop your algorithm.
4. Construct your IPO chart.

Implementation Overview

1. Start a [new python program](#) and begin implementation by adding comments in your code that document the author, the date, and the purpose of the program.
2. Type in the steps listed in the **Processing** column of the **IPO chart** as comments in your Python code.
3. Type in the Python code that actually implements each comment.

Testing Overview

1. [Compile](#) the source code.
2. Correct any errors in the source code.
3. Repeat steps 1. and 2. above until the code compiles properly.
4. Make sure the program has the correct results (output); if the results (output) are incorrect, you may need to adjust/revisit your algorithm.

To pivot towards developing a graphic-based program, we will solve the following **sample problem**:

Draw a picture of a house with two windows and a door.

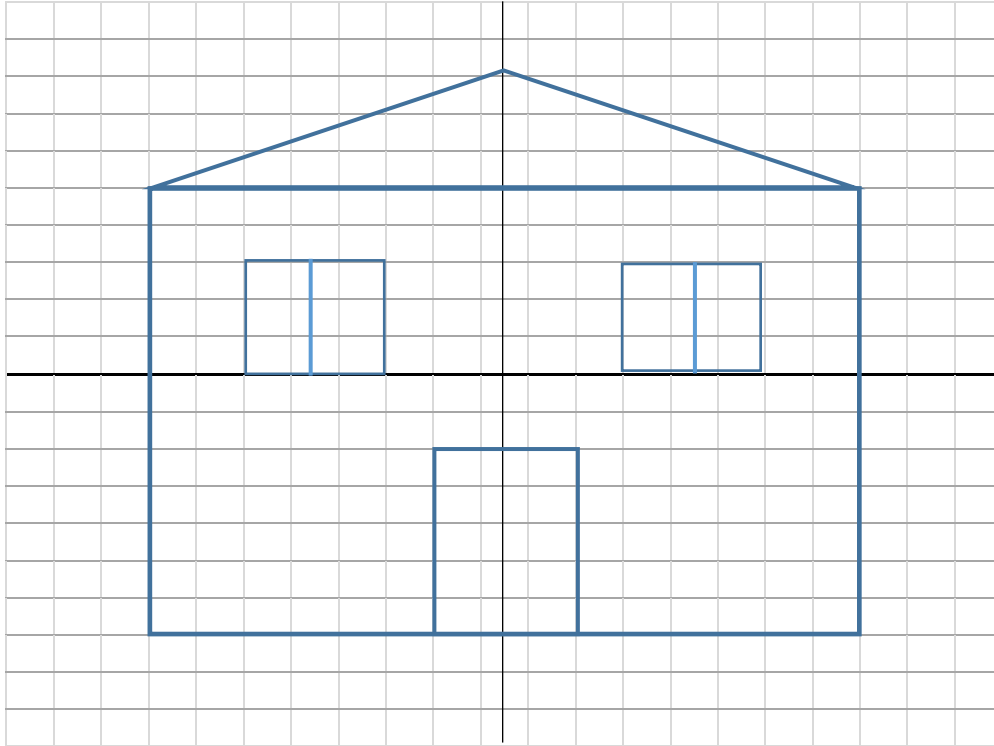
To implement the program, we will following these steps:

Planning and Algorithm Development Step-by-Step

1. **Make sure you have a clear understanding of the problem you are trying to solve.**
The driving goal of this step to ensure that we understand what the end result should be. For graphic-based programs, use this step to envision – in your mind – what the house should look like. The picture you create in your mind should have be a house with a roof, two windows, and a door. Once you have your mental picture, we're ready to move to the next step.

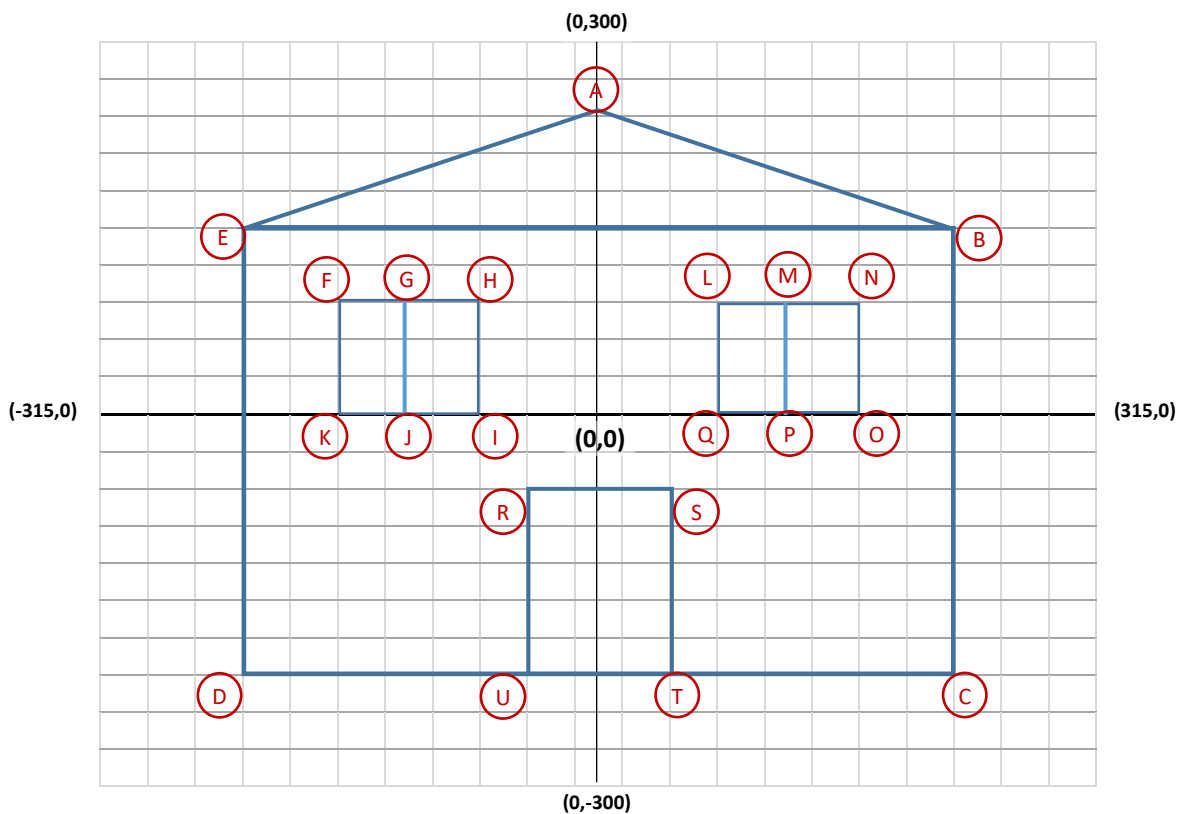
2. Draw a sketch of what the program will look like when it is running properly.

To create our initial sketch, we will use graph paper. Blank graph paper has been included in [Appendix D](#).



3. Carefully label each key point in the screen with a letter.

These letters will be used as critical reference points in the algorithm; see the example below.



Once the graph is adequately labeled, identify the coordinates of each point for later reference. **Pro Tip #8:** It may be helpful to write this out on a separate sheet of paper.

Point	Coordinates	Point	Coordinates	Point	Coordinates
A	(0,240)	H	(-75,90)	O	(165,0)
B	(225,150)	I	(-75,0)	P	(120,0)
C	(225,-210)	J	(-120,0)	Q	(75,0)
D	(-225, -210)	K	(-165,0)	R	(-45,-60)
E	(-225, 150)	L	(75,90)	S	(45,-60)
F	(-165, 90)	M	(120,90)	T	(45,-210)
G	(-120,90)	N	(165,90)	U	(-45,-210)

4. Construct your IPO chart.

Since the user will not be providing any data and no values will be calculated, the **Input** and **Output** columns of the IPO chart are empty. However, the Processing column lists each step necessary to solve the problem which, in this case, is to draw each part of the house. Read carefully through the steps below to get a feel for the flow of graphic-based code in English. Once the **Processing** column is complete, we will be ready to write our code.

Input	Processing	Output
	<ul style="list-style-type: none"> A. Import Python's drawing functionality B. Lift the pen from the drawing surface C. Draw the Roof <ul style="list-style-type: none"> a. Set the pen at point E (-225, 150) b. Lower the pen onto the drawing surface c. Draw over to point A (0, 240) d. Draw over to point B (225, 150) e. Draw over to point E (-225, 150) D. Lift the pen from the drawing surface E. Draw the Left Window <ul style="list-style-type: none"> a. Set the pen at point F (-165, 90) b. Lower the pen onto the drawing surface c. Draw over to point H (-75, 90) d. Draw over to point I (-75,0) e. Draw over to point K (-165,0) f. Draw over to point F (-165, 90) g. Lift the pen from the drawing surface <ul style="list-style-type: none"> i. Set the pen at point G (-120, 90) ii. Lower the pen onto the drawing surface iii. Draw over to point J (-120, 0) F. Lift the pen from the drawing surface G. Draw the Right Window <ul style="list-style-type: none"> a. Set the pen at point L (75,90) b. Lower the pen onto the drawing surface c. Draw over to point N (165, 90) d. Draw over to point O (165,0) e. Draw over to point Q (75,0) f. Draw over to point L (75, 90) g. Lift the pen from the drawing surface 	

	<ul style="list-style-type: none"> i. Set the pen down at point M (120, 90) ii. Draw over to point P (120, 0) 	
	H. Lift the pen from the drawing surface	
	I. Draw the Door	
	<ul style="list-style-type: none"> a. Set the pen at point U (-45, -210) b. Lower the pen onto the drawing surface c. Draw over to point R (-45, -60) d. Draw over to point S (45, -60) e. Draw over to point T (45, -210) 	
	J. Lift the pen from the drawing surface	
	K. Draw the Frame	
	<ul style="list-style-type: none"> a. Set the pen at point E (-225, 150) b. Lower the pen onto the drawing surface c. Draw over to point D (-225, -210) d. Draw over to point C (225, -210) e. Draw over to point B (225, 150) 	

Implementation Step-by-Step

1. **Start a new Python program and begin implementation by adding comments in your code that document its author, the date, and purpose.**

This may seem like a throw-away step but it is super important! It is important that you get this step into your muscle memory so that, as you get into developing more complex programs, you have a *grounding step*. The idea of grounding, in this context, is having a base step that takes the mystery of where to start out of the process. Knowing that you will always start your Python programs by creating a blank file and putting in comments that document (1) who wrote the program, (2) the date the program was started, and (3) what the code is designed to do can make the process of starting to code less intimidating.

```
# Author: [type your name]
# Date: [type today's date]
# Purpose: Draw a picture of a house with two windows and a door.
```

2. **Type in the steps listed in the Processing column of the IPO chart as comments in your Python code.**

The next *grounding step* should be to take each of the steps in your IPO chart and type them in as comment in your code.

In other words, the goal here is to get in the habit of typing in each line of your algorithm from the IPO chart into your code. The work you do now to get this basic process into your muscle memory will go a long way in making your development process more efficient when developing more complex programs.


```
# Author: [type your name]
# Date: [type today's date]
# Purpose: Draw a picture of a house with two windows and a door.

# Import Python's drawing functionality

# Lift the pen from the drawing surface

# Draw the Roof

    # Set the pen at point E (-225, 150)

    # Lower the pen onto the drawing surface

    # Draw over to point A (0, 240)

    # Draw over to point B (225, 150)

    # Draw over to point E (-225, 150)

# Lift the pen from the drawing surface

# Draw the Left Window

    # Set the pen at point F (-165,90)

    # Lower the pen onto the drawing surface

    # Draw over to point H (-75, 90)

    # Draw over to point I (-75,0)

    # Draw over to point K (-165,0)

    # Draw over to point F (-165, -90)

    # Lift the pen from the drawing surface

        # Set the pen down point G (-120, 90)

        # Lower the pen onto the drawing surface

        # Draw over to point J (-120, 0)

# Lift the pen from the drawing surface
```

```
# Draw the Right Window

# Set the pen at point L (75,90)

# Lower the pen onto the drawing surface

# Draw over to point N (165, 90)

# Draw over to point O (165,0)

# Draw over to point Q (75,0)

# Draw over to point L (75, 90)

# Lift the pen from the drawing surface

# Set the pen at point M (120, 90)

# Lower the pen onto the drawing surface

# Draw over to point P (120, 0)

# Lift the pen from the drawing surface

# Draw the Door

# Set the pen at point U (-45, -210)

# Lower the pen onto the drawing surface

# Draw over to point R (-45, -60)

# Draw over to point S (45, -60)

# Draw over to point T (45, -210)

# Lift the pen from the drawing surface

# Draw the Frame

# Set the pen at point E (-225, 150)

# Lower the pen onto the drawing surface

# Draw over to point D (-225, -210)

# Draw over to point C (225, -210)

# Draw over to point B (225, 150)
```

3. Type in the Python code that actually implements each comment.

```
# Author: [type your name]
# Date: [type today's date]
# Purpose: Draw a picture of a house with two windows and a door.

# Import Python's drawing functionality
from turtle import *

# Lift the pen from the drawing surface
penup()

# Draw the Roof

    # Set the pen at point E (-225, 150)
    setposition(-225, -150)

    # Lower the pen onto the drawing surface
    pendown()

    # Draw over to point A (0, 240)
    goto(0, 240)

    # Draw over to point B (225, 150)
    goto(225, 150)

    # Draw over to point E (-225, 150)
    goto(-225, 150)

# Lift the pen from the drawing surface
penup()

# Draw the Left Window

    # Set the pen at point F (-165,-90)
    setposition(-165, 90)

    # Lower the pen onto the drawing surface
    pendown()

    # Draw over to point H (-75, 90)
    goto(-75, 90)

    # Draw over to point I (-75,0)
    goto(-75, 0)

    # Draw over to point K (-165,0)
    goto(-165, 0)

    # Draw over to point F (-165, -90)
    goto(-165, -90)
```

```

# Lift the pen from the drawing surface
penup()

# Set the pen at point G (-120, 90)
setposition(-120, 90)

# Lower the pen onto the drawing surface
pendown()

# Draw over to point J (-120, 0)
goto(-120, 0)

# Lift the pen from the drawing surface
penup()

# Draw the Right Window

# Set the pen at point L (75,90)
setposition(75, 90)

# Lower the pen onto the drawing surface
pendown()

# Draw over to point N (165, 90)
goto(165, 90)

# Draw over to point O (165,0)
goto(165, 0)

# Draw over to point Q (75,0)
goto(75, 0)

# Draw over to point L (75, 90)
goto(75, 90)

# Lift the pen from the drawing surface
penup()

# Set the pen at point M (120, 90)
setposition(75, 90)

#lower the pen onto the drawing surface
pendown()

# Draw over to point P (120, 0)
goto(120, 0)

# Lift the pen from the drawing surface
penup()

```

```
# Draw the Door

# Set the pen at point U (-45, -210)
setposition(-45, -210)

# Lower the pen onto the drawing surface
pendown()

# Draw over to point R (-45, -60)
goto(-45, -60)

# Draw over to point S (45, -60)
goto(45, -60)

# Draw over to point T (45, -210)
goto(45, -210)

# Lift the pen from the drawing surface
penup()

# Draw the Frame

# Set the pen at point E (-225, 150)
setposition(-225, 150)

# Lower the pen onto the drawing surface
pendown()

# Draw over to point D (-225, -210)
goto(-225, -210)

# Draw over to point C (225, -210)
goto(225, -210)

# Draw over to point B (225, 150)
goto(225, 150)
```

Testing Step-by-Step

Once your code is in the interpreter check it for syntax and logical errors using the steps below.

1. **Compile the source code.**
2. **Correct any errors in the source code.**
3. **Repeat steps 1. and 2. above until the code compiles properly.**
4. **Make sure the program has the correct results (output); if the results (output) are incorrect, you may need to adjust/revisit your algorithm.**

Although the focus of this coding activity is graphics and not calculations, the basic steps for program development are largely in place. **ProTip #8**: learning the development process and committing it to muscle memory will help to ensure that you develop high-quality projects in a variety of contexts and, as the problems you address become more complex, you will have the firm foundation necessary to meet the challenges presented.

Congratulations! You have finished [Working with Turtle](#). Let's move on to the [Project](#).

Unit 8: Project

Throughout the **Software Development Learning Experience**, we have worked together to develop computer programs – starting with algorithm development, then translating those algorithms into Python, and finally testing the code to eliminate any syntax or logical errors. So far, we have applied this process twice: once to develop a program that performed a calculation and then again to develop a program that created graphics. At this point, given the experiences and resources you’ve become more familiar with as a result of this journey, you are in prime position to develop a calculation- or graphic-based program on your own. Using the process that we have followed, you are now ready to work on a project of interest to you, adding depth to your knowledge base, building muscle memory for the process, and identifying what – if any – units you may need to revisit to fill any learning gaps.

The sole purpose of this unit is to encourage you to flex your burgeoning programming skills on one of the following projects:

- **A Calculation-Based Program**
 - Develop a program that calculates the sum of 5 numbers
 - Develop a program that calculates area of a triangle
- **A Graphic-Based Program**
 - Draw a snowman
 - Draw a 3D cube

These projects were selected for their alignment with the examples we’ve worked with so far but, if you would like to develop a program outside of the ones listed, you are encouraged to do so! As a reminder, your toolset for program development includes the following:

- [Input-Processing-Output \(IPO\) Chart](#) (Unit 3)
- [The Python interpreter](#) (Unit 5)
- [Graph paper](#) and a protractor (Unit 7)

Be sure to use as many of these tools as you need and revisit the videos that accompany each unit if you need additional reminders or support.

As an important note: the suggested calculation- and graphic-oriented projects listed above aren’t excessively challenging on their own – this is by design. While you may definitely need to review your notes or do some additional research to complete each project, the driving goal of this independent exercise is for you to really get practice with the process. The more you practice what you’ve learned so far, the easier software development will be for you in the future, and the efficiency promised early on will be achieved. **Happy coding!**

Congratulations! You have finished the [Project](#). Let’s move on to the [Next Steps](#).

Unit 9: Next Steps

The units, activities, and exercises that we've completed so far were designed to introduce you to both software development and Python. Now that you have established a firm foundation, you are in prime position to expand your skills even further and build more complex programs. Recognizing the need to maintain momentum, this unit was developed to answer the following question: **are there other free resources I can use to continue growing and enriching my programming skills?**

The [Beginner's Guide to Python](#) features a host of free resources, including courses, that beginners can use to expand their knowledge base and take a deeper dive into Python and its many capabilities. **Additional free courses are listed below:**

- [Google's Python Class](#)
- [Coursera – Programming for Everybody: Getting Started With Python](#)
- [Microsoft – Introduction to Python](#)

Because these resources are free, you're encouraged to try as many as you have the time to explore. It may be helpful to learn Python from a host of different vantage points. There's a great chance that, even though you have established baseline familiarity with the language, each of these learning experiences will provide you with something unique and allow you to expand your capabilities in a different way.

One very important note: although the resources listed above can supplement your knowledge with advanced coding constructs and practices, it is important to point out that the approach presented in this series is quite unique in its focus on problem-solving for algorithm development. In particular, the heavy use of comments in this series and the idea of explicitly translating between English and Python are distinguishing factors. Other learning opportunities, such as those listed above, each have their own unique focus and perspective. Because some of the code presented in the resources listed may not have *any comments*, it may be difficult to understand what is happening in the code.

References like the [Python Standard Library](#) may help but, you may have to find additional resources and use context clues to understand the purpose of a line of code and formulate your comments in English. However, making the effort to translate each line of Python code into English is worthwhile and will advance your understanding in significant ways. In all, working through different learning approaches will deepen your knowledge, help you solve problems more efficiently, and make a cognitive link between the functionality needed to carry out tasks and the actual code necessary to implement them.

Finally, Python is one of many programming languages. There is an entire world of programming languages used in industry – including Java, C++, PHP, SQL, and HTML. Learning any or all of these programming languages will further enhance your ability to develop the skills and habits of mind that can prepare you to be a superlative software engineer. Given the syntax of Python, Java may be the next best language for you to take on. Consider the free Java course on [Code Academy](#) as a launch pad for learning the language, building proficiency, and adding to your repertoire.

Here's the key take-away: you've done a lot to build critical skills – continue to learn as much as you can! The resources listed above are a start but there are many others to consider and choose from. The key is to continue to learn by doing and watch your abilities grow as far as your interest will take you.

Congratulations! You have finished the Next Steps. Let's move on to [Reflecting on the Road Traveled and Measuring Growth \(Part II\)](#).

Unit 10: Reflecting on the Road Traveled and Measuring Growth (Part II)



Congratulations on completing the Software Development Learning Experience! Hopefully, you have learned a lot as a result of diligently completing each unit and practicing the three major steps of software development:

1. figuring out the **algorithm**, or the steps the computer will take to complete a task, **in English**,
2. translating that algorithm into a format, or **language**, the computer *understands* (in this case **Python**), and
3. testing the resulting computer program to ensure it compiles and produces the desired result.

By implementing each of the steps above in two different contexts – calculations and graphics – you have built, strengthened, and applied essential coding skills with a language – Python – that is heavily used in industry. This series allowed you the space to get better acquainted with software development, navigate the learning materials at your own pace, and connect your interest in the field with actual, tangible skills. Kudos to you for hanging in until the end!

But the journey continues. To take on evolving technological challenges, innovate, and implement forward-leaning designs as a software developer, you must continue to strive for expanded knowledge, maintenance and expansion of good programming habits, and the ability to frame and solve advanced, complex software development problems with high-quality solutions.

If you learned a lot from this experience and know a peer who might benefit from it, be sure to share these materials! As mentioned before, there is a significant gap in industry-focused K-12 software development training. It is imperative that students who might not ever gain exposure to this kind of training and support have a chance to grow their skillsets. Your help in expanding awareness of these materials will go a long way in filling that gap and, potentially, opening up new opportunities for your peers.

You've accomplished quite a bit in this series and one of the last tasks to complete is measuring your growth in understanding software development and your knowledge of key terms. On the next page, you will find the same set of questions you answered in Unit 2. The answers you provided before you started Unit 3 are effectively a snapshot of where you were. To see how far you've come, complete the questions on the next page and then compare your responses with the answers you gave before. The results of that comparison will not only provide insight into how you've advanced your understanding but they may also shed light on areas that need additional review. Celebrate the growth but, also, be mindful of where you need to improve and do the work necessary to refine your knowledge base.

In closing, thank you for taking the journey; we're excited that you took the risk to learn something new! We'd love to hear about your experience and how, from your perspective, it can be improved. We hope that your time exploring the units, learning key concepts, and doing activities was enriching and your ability to design, implement, and test code has been supercharged. Be sure to reach out to us at the link below and share your thoughts:

<https://forms.gle/t5dYQzLthehLT3aJ8>

Don't forget to track your progress and, as you continue to advance your software development skills, happy coding!

Track Your Progress: Answer the following questions to the best of your current ability – based on what you know right now. Take a look at the questions below and answer as best as you can, based on what you've learned. Compare the answers you provide below with the ones you provided in Unit 2 to see how much you have evolved over the course of the learning experience.

- 1. What are the three main processes used for developing computer programs?**
- 2. What is an algorithm?**
- 3. What is Python? What is it used for?**
- 4. In computer programming: what is an interpreter and what is it used for?**
- 5. What can you use to create graphics in Python?**
- 6. What aspects of your personality or interests might serve you well as a software developer?**
- 7. On a scale from 1 – 10, rate your ability to code in Python.**
- 8. On a scale from 1 – 10, rate your general interest in programming.**

Appendix A: Terms to Know

Learning how to develop code is as much about acquiring critical skills as it is about acquiring the language of the field. As a result, it's important that you not only focus on expanding your technical skills but your vocabulary as well. Having the right words to express yourself is critical to knowledge-building and being able to explain your thoughts, ideas, and challenges using the proper terminology.

The list of 13 terms below is not exhaustive, but it is a good start. As you discover new words, use context clues to try and understand their meaning and look them up to ensure you have the proper definition. This process not only expands understanding it expands your vocabulary.

- **Abstraction:** (*noun*) a problem-solving approach that focuses on the most essential parts of a problem and sifts out the details.
- **Algorithm:** (*noun*) a set of steps, typically used in computer programming.
- **Computational Thinking:** (*noun*) a problem-solving approach that uses abstraction and other methods to identify solutions that can be automated via an algorithm.
- **Compile:** (*verb*) process software code so that it can be executed by a computer as a program.
- **Comments:** (*noun*) parts of code that are not readable by the computer and are solely placed for human readers; typically used to document key aspects of a computer program.
- **Input:** (*noun*) in an IPO Chart, the information, tools, devices, and resources that are necessary to implement the solution.
- **IPO (Input-Processing-Output) Chart:** (*noun*) a problem-solving tool for developing algorithms that also keeps track of the resources needed to implement the algorithm and the expected outcomes.
- **Logical Errors:** (*noun*) software faults that are syntactically correct (formatted correctly) but result in errors like improper calculations or misspelled quoted text.
- **Neurons:** (*noun*) learning units that grow as the brain navigates cognitive challenges and attempts to retain and process information.
- **Output:** (*noun*) in an IPO Chart, the expected end-result(s).
- **Programming Languages:** (*noun*) Constructs used to encode algorithms in a format that both computers and humans can understand.
- **Syntax errors:** (*noun*) software faults resulting from code that is ill-formatted and/or does not conform to a set of pre-determined rules.
- **Variables:** (*noun*) used in programming, memory locations that are used in programming to store data.

Appendix B: Input-Processing-Output (IPO) Chart

Input	Processing	Output

As a reminder, the columns serve the following purposes:

- **Input:** keeps track of the information, tools, and devices that are necessary to implement the solution.
- **Processing:** keeps track of an ordered list of instructions necessary to solve the problem.
- **Output:** keeps track of the expected end-result(s).

Appendix C: The Python Interpreter

Having access to the Python interpreter and understanding how to use it are paramount to this series. This section includes critical information about downloading the interpreter and using it to both develop (type in) and execute python code.

- **Downloading, Installing, and Setting up the Interpreter**

In order to see the programs that we will develop spring into action, we must have access to a Python interpreter. You can find directions for downloading, installing, and setting up the interpreter [here](#) and a YouTube clip documenting the process [here](#).

- **Developing Programs Using Script Mode**

Python can be run in interactive mode, in which lines of code are interpreted one at a time, or **script mode**, in which a file is created with all the lines of code and run at once.

You can find a video that explains the difference between the two and how to use them [here](#). Note: **We will be using script mode in this series.**

- **Executing Programs in IDLE**

To run a Python program in script mode using IDLE, as we will do in this series, follow the steps provided [here](#).

Appendix D: Graphing Support

