<>

# Chapter 1.1: Computing

* **Code** is commands given to a computer in order to perform some task.
* A **line of code** is generally a single command.
* A **program** is a collection of lines of code that serves one or more overall functions.
* **Input** is data that is fed into a program for it to operate
* **Output** is what the computer provides in return after running some lines of code
* **Compile**ing code is translating human-readable computer code into instructions the computer can execute. In the programming flow, this functions as a check on the code the user has written to make sure it makes sense to the computer.
* **Execution** is running some code and having it actually perform its operations.
* **Control Structures** are lines of code that control other lines of code.
* Setting up:
  + The book covers four different options for writing and running code:
    - Writing files in a text editor like [Notepad++](https://notepad-plus-plus.org/) or [Sublime Text](https://www.sublimetext.com/), and running them using commands in the command line.
    - Using a desktop development environment like [PyCharm](https://www.jetbrains.com/pycharm-edu/download/) or [Netbeans](https://netbeans.org/" \t "[object Object]), which organizes both writing and running code.
    - Using a web-based development environment like [Vocareum](https://vocareum.com/" \t "[object Object]) or [Skulpt](http://www.skulpt.org/), which organizes writing and running code directly in your browser.
    - Using Python's interactive mode, which is like a fancy calculator. This is part of the [Python installation](https://www.python.org/downloads/) under the name IDLE, or can also be used through services like [Python.org](https://www.python.org/shell/) and [IPython](https://www.pythonanywhere.com/try-ipython/" \t "[object Object]).
* Additional Resources
  + [Foreword to How to Think Like a Computer Scientist: Learning with Python 3](http://openbookproject.net/thinkcs/python/english3e/foreword.html): A great write-up to some of the benefits of learning Python as a first programming language.
  + [Paul Ford, "What is Code?"](http://www.bloomberg.com/graphics/2015-paul-ford-what-is-code/): A ground-breaking article published on coding and software development that provides insights not only into the history but also the culture.
  + [Learn Python the Hard Way](https://learnpythonthehardway.org/book/): Zed A. Shaws seminal work on Python programming.
  + [The Python Programming Language](http://interactivepython.org/courselib/static/thinkcspy/GeneralIntro/ThePythonProgrammingLanguage.html), from How to Think Like a Computer Scientist
  + For additional practice through the course, check out:
  + [Code Academy's Python Course](https://www.codecademy.com/learn/python): Free, interactive coding practice.
  + [Google's Python Course](https://developers.google.com/edu/python/): Free Python lessons and exercises.
  + [CS50](https://cs50.harvard.edu/): Harvard University's Introduction to Computer course, also here on edX.
  + For other help via discussion boards and forums, check out:
  + [Stack Overflow](http://stackoverflow.com/): The place where developers ask other developers things.
  + [GitHub](https://github.com/):A collaborative repository for coding and developing.

# Chapter 1.2: Programming

* **Programming:** writing code through and iterative process of writing lines of code, attempting to execute them, and evaluating the results
* Lines of code are run in the order in which they appear
* Additional resources:
  + [The 12-year old app-developer](http://www.ted.com/talks/thomas_suarez_a_12_year_old_app_developer):Sometimes, learning to write code is just about getting started.
  + [How to Get Started Writing Code](https://www.codeschool.com/blog/2015/05/13/how-to-get-started-writing-code/):A nice simple breakdown with a few words of encouragement.
  + [The Poetry of Programming](https://www.youtube.com/watch?v=-jRREn6ifEQ): Think coding is boring? Listen to this TedTalk and think again.
  + [Anybody can Learn to Code](https://www.khanacademy.org/computing/hour-of-code/hour-of-code-tutorial/v/anybody-can-learn-code): Inspirational introduction to Hour of Code.
  + [Python Wiki](https://en.wikibooks.org/wiki/Python_Programming): A useful wikitionary for terms

# Chapter 1.3: Debugging

* **Debugging:** resolving problems in code, whether it be errors thrown in compilation or running or mismatches between the desired and observed output
* **Compilation Error:** error inherent within the code that occur during the computer’s read through of the code
* Common errors:
  + Syntax errors: code that doesn’t work with current programming language
  + Name errors: code that tries to use something that doesn’t exist
  + Type errors: code that doesn’t make sense
  + Runtime errors: errors that arise when trying to actually execute the code
    - Divide by zero: code that divides a value by zero
    - Null errors: code containing a variable that has no value
    - Memory errors: code that surpasses your computer’s memory
  + Attribute errors: error occurs when we ask for information about a variable that doesn’t make sense, like the happiness of a potato or the GPA of a turnip
* The goal of debugging is to get the information necessary to locate and fix the error
* If the problem with the code isn’t immediately obvious, try to add some code that will help make the problem clearer
* **Print Debugging (tracing):** a form of debugging where print statements are added throughout the code to check how the program is flowing
* **Scope Debugging:** a form of debugging where print statements are added to check the status of the variables in the program at different stages to see how they are changing
* **Rubber Duck Debugging:** a form of debugging where the programmer explains the logic, goals, and operations to an inanimate listener to methodically step through the code
* Advanced debugging:
  + Step by step execution
  + Variable visualization
  + In-line debugging
* Additional resources:
  + [The CS1301.com Debugging guide](http://cs1301.com/debugging/), assembled by our own Joshua Diaddigo.
  + BBC's [write-up of debugging](http://www.bbc.co.uk/guides/ztkx6sg) and more related topics.
  + The [Testing and Debugging lecture](https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-00-introduction-to-computer-science-and-programming-fall-2008/video-lectures/lecture-11/) from MIT OpenCourseware's Introduction to Computer Science and Programming.
  + How to Think Like a Computer Scientist's [lessons on debugging](http://interactivepython.org/courselib/static/thinkcspy/GeneralIntro/WhatisDebugging.html), starting there and proceeding for the next 5 lessons, as well as their [appendix on debugging](http://interactivepython.org/courselib/static/thinkcspy/Appendices/errorsAndDebug.html).

# Chapter 2.1: Procedural Programming

* **Procedural Programming**: giving instructions to the computer to carry out an order
* **Function:** a segment of code that performs a specific task, sometimes taking some input and sometimes returning some output
* **Method:** a function that is part of a class in object-oriented programming (but colloquially, often used interchangeably with function)

# Chapter 2.2: Variables

# Chapter 2.3: Logical Operators

# Chapter 2.4 Mathematical Operators