<>

# 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

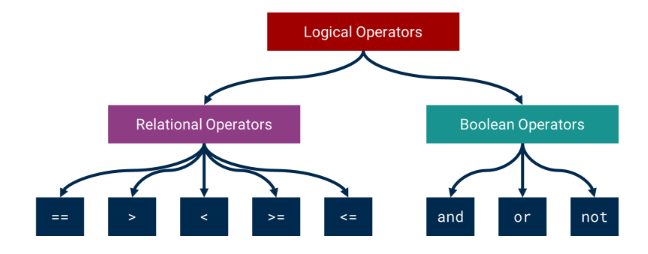
* **Programming:** writing code through and iterative process of writing lines of code, attempting to execute them, and evaluating the results
* Lines of code are run in the order in which they appear
* Additional resources:
  + [The 12-year old app-developer](http://www.ted.com/talks/thomas_suarez_a_12_year_old_app_developer):Sometimes, 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)
* **Object-Oriented Programming:** a programming paradigm where programmers define custom data types that have custom methods embedded within them
* **Event-Drive Programming:** a type of programming where the program generally awaits and reacts to events rather than running code linearly
* Logical Operators:



* **Comments:** notes from the programmer supplied in-line alongside the code itself, designated in a way that prevents the computer from reading or attempting to execute them as code
* **Documentation:** collected and set-aside descriptions and instructions for a body of code
* **Self-documenting code:** code whose variables and functions are named in a way that makes it clear what their underlying content and operations clearer to the reader
* Additional Resources:
  + [A Typical First Program](http://interactivepython.org/courselib/static/thinkcspy/GeneralIntro/ATypicalFirstProgram.html) and [Comments](http://interactivepython.org/courselib/static/thinkcspy/GeneralIntro/Comments.html) from How to Think Like a Computer Scientist
  + More on [Comments,](https://learnpythonthehardway.org/book/ex2.htm) from Learn Python the Hard Way
  + Also, some good [advice](https://en.wikibooks.org/wiki/Python_Programming/Source_Documentation_and_Comments) on comments in Python\_Programming

# Chapter 2.2: Variables

* **Variables:**  alphanumeric (letters and numbers) identifiers that hold values, like integers, strings of characters, and dates
* **Value:** the content of some variable
* **Assignment statement:** a variable is set to some value (x = 1)
* Variables are kind of like questions, and values are kind of like their answers. The variable sticks around, while the value changes.
* We always assign values to variables
* Null is the value for any variables that hasn’t been assigned a value otherwise
* **Null:**  the “value” a variable has when it doesn’t actually have a value
* In Python we uses the word “assign” synonymously with “give”. To assign a value to a variable is to give a value to a variable
* When Python hits an error, it quits right there and doesn’t continue trying to run the code, unless we handle the error.
* **Data type:** the type of content a variable holds, like an integer or a string of characters
* Basic data type:
  + Integers, or whole numbers
  + Real numbers, or numbers that can have decimals
  + Characters, like individual letters and numbers
  + Strings, which are collections of characters in a row
  + Booleans, which just hold either true or false
* Converting data types:
  + str(variable): Takes as input some variable and returns a string representation of the variable’s value. Every data type can be converted to some kind of string.
  + int(variable): Takes as input some variable (usually a string) and attempts to convert it to an integer, returning the integer if successful or raising a ValueError if unsuccessful. This function will work if variable is a string made up only of digits and, optionally, the negative sign.
  + bool(variable): Takes as input some variable (usually a string) and attempts to convert it to a boolean, returning the boolean value if successful or raising a ValueError if unsuccessful. Generally, this function returns False if variable is 0 or an empty string, True if variable is anything else.
  + float(variable): Takes as input some variable (usually a string) and attempts to convert it to a float, returning the float if successful or raising a ValueError if unsuccessful. This function will work if variable is a string made up only of digits and, optionally, a negative sign and a decimal point.
* Additional Resources:
  + How to Think Like a Computer Scientist's descriptions of [Expressions](http://interactivepython.org/courselib/static/thinkcspy/SimplePythonData/intro-VariablesExpressionsandStatements.html), [Data Types](http://interactivepython.org/courselib/static/thinkcspy/SimplePythonData/ValuesandDataTypes.html), [Type Conversions](http://interactivepython.org/courselib/static/thinkcspy/SimplePythonData/Typeconversionfunctions.html), [Variables](http://interactivepython.org/courselib/static/thinkcspy/SimplePythonData/Variables.html), [Reserved Words](http://interactivepython.org/courselib/static/thinkcspy/SimplePythonData/VariableNamesandKeywords.html), and [Input](http://interactivepython.org/courselib/static/thinkcspy/SimplePythonData/Input.html).
  + The Hello World Program's [write-up on naming conventions](http://www.thehelloworldprogram.com/python/python-variable-assignment-statements-rules-conventions-naming/).
  + The Python Programming Wikibook's [description of data types](https://en.wikibooks.org/wiki/Python_Programming/Data_Types).
  + Panicked? Don't be. The [Zen](http://python.net/~goodger/projects/pycon/2007/idiomatic/handout.html) of Python

# Chapter 2.3: Logical Operators

* **Operators**: specific, simple functions that act on primitive data types, like integers and strings
* **Modulus:**  remainder function. The remainder when one integer is divided by another
* **Mathematical Operators:**  operators that perform mathematical functions, like adding numbers together or assigning values to variables
* **Logical Operators:** operators that perform logical operations, such as comparing relative values, checking equality, checking set membership, or evaluating combinations of other logical operators
* Ultimate goal of all logical operators is to assess whether certain statements are true or false
* Two kinds:
  + Relational operators: check if things are true
  + Boolean operators: check the combination of multiple relational operators
* **Numeric Comparison Operators:**Operators that facilitate numeric comparison between values. Typically, these are 'greater than' (>), 'greater than or equal to' (>=), 'equal to' (==), 'less than' (<), and 'less than or equal to' (<=).
* **Non-Numeric Equality Comparisons:**Nearly any kind of data can compare for equality, even if it isn't numeric. We can't ask if an apple is greater than an orange, but we can ask if apples and oranges are 'equal', or the same thing. In practice, sometimes this will compare values to see if the values of two variables are the same, and other times it will compare if two variables are pointing to the same data in memory. It could be that two variables have the same values, but those values are stored in separate places, and so the computer doesn't recognize them as equal.
* **Set Operators:**Check to see if a value is a member of a set of multiple values. Most often this comes up in strings and lists.
* **= is the assignment operator**
* **== is for comparison**
* **Boolean Operators:** Operators like “and” and “or” that act on pairs of boolean (true or false) values, or that act on single boolean values, like “not”.
* **And:** An operator that acts on two boolean (true or false) values and evaluates to “true” if and only if both are true.
* **Or:**An operator that acts on two boolean (true or false) values and evaluates to “true” if and only if at least one is true.
* **Not:** An operator that acts on one boolean (true or false) value and evaluates to the opposite value (false becomes true, true becomes false).
* **Truth Tables:** Tables that map out the results of a statement in boolean logic (that is, using boolean operators) depending on the values of the individual variables.
* **Communicative Property:**  the order in which variables are presented does not matter if only one type of operator is used
* **Distributive Property:** while comparing “a and (b or c)” with “(a and b) or (a and c)”, distributing the “a and” to the individual operators inside the parentheses give the same result.
* **Additional Resources:**
  + [Statements and Expressions](http://interactivepython.org/courselib/static/thinkcspy/SimplePythonData/StatementsandExpressions.html) and [Operators and Operands](http://interactivepython.org/courselib/static/thinkcspy/SimplePythonData/OperatorsandOperands.html), from How to Think Like a Computer Scientist
  + [Boolean Values and Boolean Expressions](http://interactivepython.org/courselib/static/thinkcspy/Selection/BooleanValuesandBooleanExpressions.html),[Logical operators](http://interactivepython.org/courselib/static/thinkcspy/Selection/Logicaloperators.html), and[Precedence of Operators](http://interactivepython.org/courselib/static/thinkcspy/Selection/PrecedenceofOperators.html), from How to Think Like a Computer Scientist
  + [Memorizing Logic](https://learnpythonthehardway.org/book/ex27.html), from Learn Python the Hard Way
  + [Boolean logic](http://www.thehelloworldprogram.com/python/python-boolean-logic-not-as-scary-as-it-sounds/), from The Hello World Program

# Chapter 2.4 Mathematical Operators

* **Mathematical operators**: take one or more numbers as input and produce usually a number as output
* **Assignment Operator:** an operator that takes the output of an expression and assigns it to a variable
* Modulus is the remainder function. When you divide two integers if one does not go evenly into the other, you have a remainder. E.g. 3 goes into 7 twice with 1 left over. So 7 modulus 3 would give a result of 1. It returns only the remainder of the division, not the result itself.
* How do you check if a number is even? <number> modulus 2 is 0. If there is no remainder when the number is divided by 2, then it’s an even number.
* **Floor Division**: represented by a double-slash (//), which is division that rounds down to the nearest whole number. E.g. 5 //2 would be 2, because 5 divided by 2 is 2.5, which is rounded down to the nearest integer, 2.
* **Exponentiation**: represented by a double-asterisk (\*\*), which raises the first number to the second number. For example, 5 \*\* 2 would be 25, because 5 raised to the 2nd power is 25.
* Modulus operator executes the remainder function. It divides two integers, but instead of returning the quotient, it returns the remainder of the integer division
  + Every even number when divided by 2 gives the remainder, and every odd number when divided by 2 gives a remainder of 1
* **Self-Assignment**: a common programming pattern where a variable is assigned to the output of an expression that included the variable itself
  + new\_balance = old\_balance + paycheck
  + balance = balance + paycheck
* **Increment:** repeatedly adding a constant, typically one, to a variable.
* Any time you combine integers and floats the result will be a float