* Download and set up VS code and git
* Getting into Apex
  + Go to developer console
  + File extension is .apxc
  + Saving cannot be done if there are errors
  + Be sure to save often
    - This is why we generally use VSCode
  + What is it
    - Is the programming language used within salesforce
    - Hosted
      * Compiled in the cloud
    - Strongly typed
      * The language itself cannot convert between unrelated data types
    - Multi-tenet aware
      * Is aware of the resources it is using along with other orgs
      * It is designed to run efficiently with other tenets
      * Languages built-in support for inline DML, SOQL, SOSL
    - Object oriented
      * Operations are performed on and with objects
    - Case-insensitive
      * Uppercase and lowercase are treated the same
      * Still try to keep to a convention
        + Camel case is used at Revature
    - Proprietary
      * Salesforce decides how and where we can use it
      * The lightning platform is the only place it can run
    - Java-like
      * Not based on java, only similar in syntax
  + Used when you need something more powerful than a declarative programming language
  + House rules
    - Not required but recommended
    - Use comments
    - Use a header
      * Name of project
      * Author
      * Created
      * Last modified
      * Description
    - Classes and methods use upper camel case
      * Starts with capital letter on first one too
    - Variable names are lower camel case
  + Variables
    - We use variables to store values to be referenced and used somewhere else
    - Make variables self documenting
    - Types
      * Blob
        + Holds binary data
      * Boolean
        + Uninitialized are null
      * Date
      * Datetime
      * Decimal
        + Arbitrary based on its declaration but can store non-whole numbers
      * Double
        + 64 bit version of decimal
      * Id
      * Integer
        + 32 bit size
      * Long
        + 64 bit version of integer
      * Object
        + Definition depends on object
        + Refers to an instance of a class
        + Can also reference standard and custom objects in salesforce
      * sObject
        + Reference a standard or custom object in salesforce
        + This also applies to API names
      * String
        + Can only use single quotes
      * Time
    - Collections
      * Non-primitive data types
      * Are divisible (can be broken down to smaller types)
      * List
        + A list of things (duh)
        + Can grab a piece from a list
        + Comma separated
        + Ordered
        + Indexed
        + Single values are called elements
        + All share the same data type
        + Instantiation

List<account> listDataType = new List<Account>();

* + - * Sets
        + Unordered collections
        + Must contain unique values
        + Instantiation

set<sObject> setDataType = new Set<sObject>();

* + - * Map
        + Group of key value pairs
        + Each key must be unique
        + Can have different values
        + Can have null keys but try not to for best practice
        + Instantiation

map<String, Boolean> mapDataType = new Map<String, Boolean>();

* + - Enums
      * Enumeration
      * Not as common
      * Abstract data type
      * Have pre-defined values
      * Don’t adherer to numeric order
      * Instantiation
        + Public enum Season(WINTER, SPRING, SUMMER, FALL);
        + Season currentSeason = Season.WINTER;
      * No spaces or string literals
      * Used less than lists due to the lack of oob functionality
  + Initialization blocks
    - Execute prior to the constructor
    - Only run once per object
    - Should be reserved for complex situations
  + Static blocks
    - Only execute once when the class is first accessed
  + Constructors
    - Code that is invoked when an object is instantiated
    - Always has a no argument constructor by default
      * When we create our own, this is removed
  + Methods
    - Actions to be performed on or with an object
    - Represent the behavior of the class
    - Contain logic to result in a specific outcome
    - To declare
      * accessModifier returnType methodName(parameters){
      * logic
      * }
    - Example
      * Public void WriteDebug(string target){
      * hello = target.helloMethod();
      * system.debug(hello);
      * }
    - Parameters are passed by reference generally, but primitives are passed by value
  + Control statements
    - If
      * If(character == ‘Darth Vader’){
      * System.debug(‘This is Luke’s father’);
      * }
    - Else
      * Put { right after } from the if
    - Else-if
      * }else if(blah == blah){…
    - While loop
      * Check a condition and repeat until the condition is false
      * Same syntax as everywhere else
      * Do while also exists
    - Switch statement
      * Switch on varName{
      * When logic1{…..}
      * When logic2{…..}
      * …
      * }
    - For loop
      * Executes a set amount of times
      * for(integer i = 0; i < Example.size(); i++){…
      * List/set for loop
        + Executes for every value that is found in the list that is placed in it
        + for(integer i:Example){…