**Sunday**

**Evening Class: Introduction** (120 minutes)

Objective: Coders will get comfortable with each other, the instructor, and the format of the course. Coders will understand the basics of using the command line and git. A review of basic concepts needed for the course will be included.

* (~15 minutes) Encourage the coders to get to know each other with a few ice breakers.
* (~5 minutes) Discuss why they are here and write down goals for the week.
* (~5 minutes) Give the students an idea of the schedule for the week.
* (~10 minutes) Classroom expectations: Let the students know what you expect from them - raising hands, being respectful to one another, collaborative not competitive, etc. Expectations on the computer: staying on the site the teacher tells you to be on. Encourage students to ask questions when they hear words they don’t know or get lost during a presentation.
* (~10 minutes) Command line basics: Introduce commands ls, cd, mkdir, rmdir, rm, nano, clear, javac, and java.
* (~10 minutes) Command Line Practice: see page 1 of assignments. Have them replicate that structure on their desktops using only the command line.
* (~15 minutes) Github setup and basics
  + Have all coders make a github account here: <https://github.com/>
  + Walk them through creating a local repo, creating and committing a file, and creating a Github repo and connecting it to the local, and pushing changes to github - details are in source and the student manual if needed. Make sure they understand that all projects in the class should be in gibhub, and that all commits should have an appropriate commit message.
    - Through this, coders should be familiar with the following commands: git init, git status, git add, git commit -m “your message here”, git push origin branchname, and git branch
* (~30 minutes) Review of Basic Concepts
  + Object-Oriented Programming, Java Syntax
  + Defining classes, variables, and methods
  + Basic I/O
  + Polymorphism
  + Test Driven Development and Junit
* (~20 minutes) Netbeans IDE Tour. Guide students through creating a basic “Hello, World” application using the Netbeans IDE
  + New->project, name it “HelloWorld”, put it in the git projects folder.
  + Add file->Java main class, name is “HelloWorld”.
  + Where it says “// TODO code application logic here”, replace with “System.out.println(“Hello,World”);”.
  + Press the green arrow to run
  + Save file and add and commit to Git
  + Give students time to poke around Netbeans. Share this shortcuts list: <https://shortcutworld.com/en/NetBeans/8/win/all>
  + Example of finished project is in HelloWorld project

**Monday**

**Morning Class: Linked Lists** (180 minutes)

Objective: Coders will learn about the concept of Data Structures. They will learn about linked lists, analyzing an example of a linked list implementation and creating their first project implementing a doubly linked list in Java.

* (~10 minutes) Introduce the idea of data structures: ways of organizing elements on a computer. Examples of types of data which should require different data structures:
  + Students in a class (want to be able to sort by different attributes)
  + Customers waiting for a table at a restaurant (FIFO - first in, first out)
  + Steps taken in a maze (LIFO - last in, first out)
* (~5 minutes) Arrays in Java: An indexed list of elements, all of which have the same type. Explain how to create, populate, and access elements of an array. Go over ArrayExample.java in the In Class Examples folder.
* (~15 minutes) ArrayList in Java: Java has a List type that has several implementations. Most frequently used is the ArrayList class, which has a flexible size. Explain how to create, populate, and access elements of an ArrayList. Go over ArrayListExample.java in the In Class Examples folder, explain each part
* (~45 minutes) Linked List in Java: In a singly linked list each node in the list stores the contents of the node and a pointer oto the next node in the list. It is called a singly linked list because each node only has a single link to another node. To store a single linked list, you only need to store a pointer to the first node in that list. The last node has a pointer to nothingness to indicate that it is the last node.
  + Go over the SinglyLinkedListImpl class in the LinkedListImplementation project. Have coders help explain each method. Then go over the example (in the main part of that class) and have coders guess what will happen. Ask coders to help modify the example and anticipate different outcomes.
  + First unit testing: Go to the Test Packages and open the Linked List Test. Show codes how junit tests work. Have them help write a few more to test different methods in the Linked List Implementation.
* (~15 minutes) Go over doubly linked lists. How are these useful?
* (~90 minutes) Have coders do the Doubly Linked List project.

**Afternoon Class: Stacks** (180 minutes)

Objective: Coders will become familiar with the concept of Stacks and create a project implementing a stack in Java.

* (~20 minutes) Stacks: Have two principal operations: *push*, which adds an element to the collection, and *pop*, which removes the most recently added element that was not yet removed. Additionally, apeek operation may give access to the top without modifying the stack.
* (~10 minutes) Show the coders how to check for matching brackets using a stack. Have coders do the Bracket Matching Worksheet.
* (~10 minutes) Reverse Polish notation is a mathematical notation in which every operator follows all of its operands, in contrast to Polish notation (PN), which puts the operator before its operands. It is also known as postfix notation. Examples:
  + 3+4 is written as 3 4 +
  + 5-2 is written as 5 2 -
  + (3 - 4) x 5 is written as 3 4 - 5 x or 5 3 4 - x
* (~10 minutes) Have coders do the Reverse Polish Notation Worksheet
* (~130 minutes) Students work on the Stack Calculator Project - implementation of an RPN calculator using a stack.

**Evening Class: Seminar 1** (120 minutes)

Objective: Students will be introduced to the programming language Python. They will do a tutorial in Python, and compare and contrast Python with Java.

* (~5 minutes) Introduce Python: Python is an interpreted, high-level programming language. It used general-purpose, so it can be used for a lot of different types of projects. Python differs from many other languages in its use of whitespace instead of punctuation, and its extremely efficient and succinct syntax.
* (~60 minutes) Have the coders work through the Python tutorial here: <https://www.codecademy.com/learn/python>, as much as they can get through in the half hour.
* (~10 minutes) Discuss how Python is similar to Java. Discuss common features in syntax, how the languages are similarly object-oriented, and other similarities.
* (~15 minutes) Discuss the differences between Python and Java. There are many differences in syntax, and in the purpose and common uses. Discuss the pros and cons of using each language.

**Tuesday**

**Morning Class: Sorting Algorithms** (180 minutes)

Objective: Coders will be exposed to several sorting algorithms. They will be able to see one implementation, and create one themselves.

* (~5 minutes) Intro to sorting: Why is sorting important? If you have a list of data, being able to get it in a certain order is often necessary
  + Yelp: sorting by proximity and popularity
  + Students in a class: sort in alphabetical order
  + Manual sorting: if I give you a list of numbers, how do you sort it from smallest to largest?
    - Pick the smallest number, write that down, repeat until no more numbers
    - If it’s a really long list, sort smaller lists and then combine them by comparing smallest numbers
* (~10 minutes) Look at this sorting algorithm animation: <https://www.toptal.com/developers/sorting-algorithms>
  + Not all algorithms work at the same speed, and some are better with different initial setups
  + Which of these look the most efficient?
  + Which will be covered? Bubble, selection, insertion, merge (later in the week)
* (~45 minutes) Talk through implementations of Bubble and Selection sort in the In Class Examples project called SortingExamples.
* (~90 minutes) Have coders do the Sorting Time Comparison project.

**Afternoon Class: Queues** (180 minutes)

Objective: Coders will become familiar with the concept of queues and create their next project implementing a queue in Java.

* (~15 minutes) Introduce queues to the coders. Go over the concept of first in, first out and the enqueue and dequeue operations.
* (~15 minutes) Go over the array implementation of a queue in the In Class Examples folder.
* (~150 minutes) Have coders do the Queue project.

**Evening Class: Seminar 2** (120 minutes)

Objective: Students will be introduced to the programming language C. They will do a tutorial in C, and compare and contrast C with Python and Java.

* (~5 minutes) Introduce C: C is an imperative programming language with a static type system. C has a memory-use system that uses pointers, and is very fast. C is often used in operating systems, comilers, editers, databases, and database management systems.
* (~60 minutes) Have the coders work through the C tutorial here: <http://www.learn-c.org/> , as much as they can get through in the half hour.
* (~10 minutes) Discuss how C is similar to Java and how it’s similar to Python. Discuss common features in syntax and other similarities.
* (~15 minutes) Discuss the differences between C and Python and Java. There are many differences in syntax, memory access, and in the purpose and common uses. Discuss the pros and cons of using C versus Python and versus Java.

**Wednesday**

**Morning Class: Advanced Sorting Algorithms** (180 minutes)

Objective: Coders will learn about and implement advanced sorting algorithm MergeSort, as well as binary search on sorted lists.

* (~10 minutes) Introduce the concept of Merge Sort: dividing a list in halves repeatedly, and then merging all the pieces together until the list is sorted.
* (~5 minutes) Write down random numbers on twice as many pieces of paper as there are students. Have them perform the merge sort algorithm to sort the paper.
* (~5 minutes) Explain binary search: A very efficient way of searching in a sorted list. Binary search works by reducing the size of the search space by half repeatedly until you find your query or ensure it is not in the list.
* (~160 minutes) Have coders do the Merge Sort Project. Be aware that this is a very hard project and they will likely all struggle with it.

**Thursday**

**Morning Class: Trees and Heaps** (180 minutes)

Objective: Coders will learn about trees, binary trees, heaps and heapsort.

* (~30 minutes) Introduce the concept of trees, and binary trees, and walk through the In Class Example Tree and TreeTest, which constructs a tree of the line numbers of all the words in a document and uses that to print them in alphabetical order.
* (~60 minutes) Have coders do the Tree project.
* (~20 minutes) Introduce Heaps: A heap is a complete binary tree, filled from left to right. A min-heap has the property that the value of all parents is less than or equal to their children, so the root will be the minimum. A max-heap has all parents with values greater than or equal to their children, with a maximum root.
* (~10 minutes) Introduce heap sort: A process by which data is sorted by creating a heap and repeatedly picking off the top node(min or max) and heapifying the remaining data
* (~60 minutes) Have coders do the Heap Sort project.

**Afternoon Class: Begin Final Projects** (180 minutes)

Objective: Coders will be given their final assignment for the week: A choose-your-own-adventure game. They have 2 full class periods to work on it.

* (~180 minutes) Have coders start the Choose Your Own Adventure project. Demonstrate the final functionality in the Solution before they start so they know what they’re aiming for.

**Evening Class: Seminar 3** (120 minutes)

Objective: Students will be introduced to the programming language C++. They will do a tutorial in C++, and compare and contrast C++ with C, Python, and Java.

* (~5 minutes) Introduce C++ (Pronounced See Plus Plus): C++ is built off of C, so it is compatible with most C programs. It’s a compiled language, and can be one of the fastest languages when used correctly. It is an object-oriented language. Like C, C++ has low-level memory manipulation (pointers).
* (~60 minutes) Have the coders work through the C++ tutorial here: <http://www.learncpp.com/>, as much as they can get through in the half hour.
* (~10 minutes) Discuss how C++ is similar to C, Java, and Python. Discuss common features in syntax, which languages are similarly object-oriented, and other similarities.
* (~15 minutes) Discuss the differences between C++ and C, Python, and Java. Discuss the pros and cons of using C++ versus C, versus Python and versus Java.

**Friday**

**Morning Class: Continue Final Projects** (180 minutes)

Objective: Coders will continue working on thir final projects with the intention of finishing during this period.

* (~180 minutes) Coders work on final projects. If they finish early, encourage them to add more exciting features to their projects. If they are finished quite early, have them learn about hashmaps.

**Afternoon Class: Finish Final Projects and Prepare Presentations** (180 minutes)

Objective: Coders will finish their final projects if they have not yet. Then they will prepare and practice their final presentations.

* (~5 minutes) Today, we will be making and practicing our final presentations for families which will happen tomorrow.
* (~60 minutes) Coders work on their presentations.
* (~45 minutes) Coders break into small teams and practice presenting each other and giving constructive feedback.
* (~70 minutes) Coders can continue to work on their presentations, put finishing touches and additional features on their projects, or work on a new project.

**Evening Class: Seminar 4** (120 minutes)

Objective: Students will be introduced to the programming language Racket. They will look through a Racket tutorial, and compare and contrast Racket with Python, C, C++, and Java.

* (~5 minutes) Introduce Racket: Racket is a multi-paradigm programming language. It is related to well-known languages Lisp and Scheme. It is used as a platform to develop other languages, for scripting, and for graphics. It is considered a great learning language.
* (~60 minutes) Look over the Racket tutorial with the coders, as much as you can get through in the half hour.
* (~10 minutes) Discuss how Racket is similar to C, C++, Java, and Python. Discuss common features in syntax, which languages are similarly object-oriented, and other similarities.
* (~10 minutes) Discuss the differences between Racket and C++, C, Python, and Java. There are many differences in syntax, and in the purpose and common uses. Discuss the pros and cons of using Racket versus C++, C, Python, and Java.
* (~5 minutes) Discuss why Racket would be a good language to learn first when starting out in computer science.

**Sources:**

<http://product.hubspot.com/blog/git-and-github-tutorial-for-beginners> for Github directions in Student Manual

<https://shortcutworld.com/en/NetBeans/8/win/all> Netbeans shortcuts

<https://www.toptal.com/developers/sorting-algorithms> sorting algorithm animations

<http://beginnersbook.com/2013/12/java-arraylist/> in class examples -> arraylist example

<http://www.java2novice.com/data-structures-in-java/linked-list/singly-linked-list/> in class examples -> linked list example

<https://en.wikipedia.org/wiki/Stack_(abstract_data_type)> definition

<https://en.wikipedia.org/wiki/Reverse_Polish_notation> definition & image in manual

<https://learnxinyminutes.com/docs/racket/> racket tutorial

<http://www.java2novice.com/java-sorting-algorithms/> sorting algorithms

<https://www.tutorialspoint.com/data_structures_algorithms> sorting algorithm descriptions

<http://www.sanfoundry.com/java-program-implement-heap-sort/> heapsort implementation