Mid-Term Review

The purpose of this document is to help you prepare for the midterm exam in CSC 100. This is something that you should do for any test you take in any class: Go through the material that is covered, and outline a list of topics that have been covered. It is generally your responsibility to do this, and usually it's not something instructors will provide for you. Regardless of whether you are given a list of topics, it's a good exercise and memory refresher to scan through the original materials (your notes, outside of class readings, any provided lecture slides/materials, lab write-ups, etc.) and summarize what has been covered. A good exercise for this class would be to set this handout aside, go through the material yourself and make a list, and then compare with my list. This will help you see if you have a good picture of the topics that are important – is anything missing from your list?

The midterm consist primarily of short answer questions, where short answer questions can involve Snap! programming topics, and there will be some terminology questions as well. You might be asked to sketch out a Snap! script to solve a particular problem, or to describe what a provided Snap! script does. If you are asked to sketch out a script you should use the "writing style" that was handed out with Homework 2, so it would be worthwhile to practice writing things out like this before you come to the exam! The main purpose is to clearly express your understanding of algorithmic solutions, not to demonstrate any memorization of Snap! blocks. For example, if you are sketching out a script that inserts an item in a list, you would write that out something like "insert (player) at 1 of (player list)". What's not very important is the precise wording (did you use "at" or "in", etc.). What *is* very important is that you demonstrate something that makes sense as far as the underlying concept of inserting something in a list. Inserting something in a list must include three parameters: what you are inserting in the list, where you are inserting it in the list, and what list you are inserting into. It simply doesn't make sense to talk about inserting something into a list unless you give those three pieces of information. The precise wording of how you express that is much less important than the concept that it represents, and how that reflects your mental model of what an insertion into a list does.

The rest of this document is broken down into the three main sources of content in this class: The lectures, *Blown to Bits* readings, and lab exercises. Once you have a list of topics (my list or your list), you should go through it carefully and think: "Do I remember what was covered on this topic, and do I understand it?" "What kinds of questions might be asked about this topic, and do I know how to answer them?" Even better would be a study group where you go through the list of topics and some people in the group make up questions for topics, and the others propose answers (and then rotate roles in the group).

Lecture Topics

While all topics are important (remember that!), one particularly useful strategy would be to look through the lecture slides and find the places where specific questions were posed to the class. If you were asked to solve some problem in class, then that is an indication that it might be a good test question!

Lecture 1 and 2: Class Introduction and Success in College

- Note: None of this is content for the exam

Lecture 3: Introduction to Computer Science

- Assigned readings: “Coding is coming...” and “A commencement speech...”
- Basic definition and history – especially the general meaning of “computer” and “computer science”
- Euler's GCD algorithm (when? why important?)

Lecture 4: Computing and Programming

- Basic computational step – what is a step, and what isn't
- Role of programming languages – high level vs. low level
- Software engineering – waterfall model
- Pair programming idea – roles, how to work, etc.

Lecture 5: Abstraction

- General principles: detail removal and generalization
- Using Snap! blocks (or functions and procedures) for abstraction

Lecture 6: Data Representation - Part 1, Numbers, Bases, and Binary

- Number representations and bases
- Bases: Decimal and binary
- Converting between decimal and binary (both directions)

Lecture 7: Data Representation - Part 2, Hexadecimal and Practical Issues

- Another useful base: hexadecimal
- Converting between hexadecimal and decimal (both directions)
- Converting between hexadecimal and binary (both directions)
- Use of hexadecimal in file dumps

Lecture 8: Organizing Data – The power of structure

- Abstract Data Types
- Data Structures
- Lists
- Dictionaries and trees

Lecture 9: Algorithms - Part 1, The Basics

- Definitions and concepts
- Problems vs algorithms
- Algorithm properties and trade-offs
- Computing speed, Moore's Law, and the singularity

Lecture 10: Algorithms - Part 2, Measuring Time

- What does “time” mean for an algorithm?
- Why “stopwatch time” isn't a good measure
- Notion of “steps” (similar to Lecture 4)

Lecture 11: Algorithms - Part 3, Time Complexity Basics – Constant, Linear, and Quadratic Time

- Constant time
- Basic loop patterns and time complexity
- Linear time
- Quadratic time
- Estimating running time of code using algorithm complexity

Lecture 12: Algorithms - Part 4, More Time Complexity – Logarithmic and Exponential Time

- Searching in a sorted list
- Logarithmic time
- Exponential time
- Relation between linear, quadratic, logarithmic, and exponential time

Readings

The class readings cover just a few core topics of how technology is affecting the world and our daily lives, and the impact of that on personal privacy and other issues. The core topics are illustrated with a lot of individual stories. While the individual stories are not so important for the factual details they contain, they are important as good illustrations of the core topics, so make sure you know the basics of these stories. You might be asked, for example, to describe an instance in which a computer user leaves "fingerprints" from their actions (and you could then describe one of the following stories: the person who took pictures of the last Harry Potter book which had the camera's serial number embedded, the story about the rental car that was taken into Nevada when only California was authorized, or information about a color printer leaving subtle dots on each page it prints, …).

Blown to Bits: Chapters 1-2; and the "Emma" story (Only the high points are listed here)

- Prevalence (ubiquity!) of digital data, and information represented in digital form (bits)
- Exponential growth of computing power and data capacity over time
- Digital "footprints" and "fingerprints"
- RFID tags and other ways of tracking people or things
- Controlling Parkinson's tremor with a device (Emma)

Lab Exercises

The labs give you experience with Snap! and (some) Python programming, but also introduce some big concepts that are important well beyond just getting the lab done. While you need to be familiar with Snap! programming, and you need to be able to sketch out programs and describe how Snap! scripts work, the most important thing with the labs is to have a strong understanding of the big concepts and how they apply not only in that particular lab but in general. The other tip with the labs is to go through the lab quizzes - these are designed to ask questions about the most important concepts in the labs. If you understand the answers to these questions (really understand why the answer is what it is, not just that you can repeat the answer), then you understand a lot of the main points in the labs. Those quizzes aren't a comprehensive coverage of important topics, of course, but they're a good start! You should also pay attention to terminology, which is generally bold and underlined in the lab exercises, and you should be comfortable with all of these words and phrases!

Lab 1: Introduction to Scratch/Snap! - animations and communication

- Snap! concepts - sprites, scripts, costumes, etc.
- Blocks, parameters, and arguments
- Events and broadcast messages
- Big concepts: Problem decomposition, event-driven programming, concurrency, and versioning

Lab 2: Drawing, Interaction, and Variables

- Turtle graphics and drawing
- Sprite movement and smooth animation
- Use of variables in Snap! scripts
- Loops and iteration

Lab 3: Abstraction with Functions

- Types of Snap! blocks: command (which can be regular or C-blocks), hat, and reporter (which can be a predicate or a general reporter)
- Important terminology: function, procedure, parameters, reporting, and returning
- Data types: numbers, strings, and Boolean values

Lab 4: Number representations, strings, and Unicode

- Character encodings, converting characters to codes and vice-versa
- Number base conversions
- for loops
- Processing strings character by character
- Describing what a block does at a high level

Lab 5: Using lists for data

- Basic list concepts and operations: retrieving items, inserting/adding items, deleting items, replacing items, and checking whether a list contains an item
- Creating lists (the range function)
- Processing lists an item at a time using for loops
- Generating and using random numbers
- Shuffling a list
- Adding up items in a list

Lab 6: Introduction to Python

- Variables in Python
- Python turtle graphics and differences with Snap!
- The Python for loop and the range function
- Defining functions in Python and returning values
- Conditionals in Python (if, if/else, and if/elif)
- Strings vs integers in Python
- Getting input in Python
- Lists in Python

Lab 7: Algorithms

- Testing algorithms and measuring time (Python and Snap!)
- Predicting running times
- Sorting a list