Exam 2 Study guide

(Note: this is mostly Dr. Johnson's study guide for the majors' sections. When I read it, I realized - hey, this is pretty darned applicable for our section, too. I made a few changes, but not many. Giving credit where credit is due.)

This study guide covers *the material you are responsible for on the second exam*. This guide is expressed in terms of skills you must be able to demonstrate on the exam. You are going to be expected to evaluate, write, and examine Python code to demonstrate the skills covered in this study guide. Additionally, there will be "normal" questions (multiple choice, short answer, etc.) where you will demonstrate your understanding of the material.

Exam 2 is similar to Exam 1, except that it is worth more points. Exam 2 will be administered on Blackboard, and will be an open book/open notes exam. You will have 90 minutes to complete the exam. I will try to put a "practice exam" on Blackboard ahead of class so that, if you have never had the experience of taking a test on Blackboard, you can get some practice.

Computer science is by nature ***cumulative***. Thus, concepts from the first exam will be present on this exam, though they will not be the focus. You're not allowed to forget how a "for each" loop works

The order, length or depth of a section in this study guide is ***no indication*** *of the relative importance of that topic or its likelihood to appear on the exam.* i.e. Don't ignore the short sections!

While Loops

- *Define* sentinel values, particularly in terms of constants
- *Define* boolean flags and explain their use in code
- Given a loop, *identify* any sentinel values or boolean flags in use

Constants

- *Define* magic numbers and *explain* why they are undesirable, in terms of:
    - Code readability (how easy it is to read)
    - Code maintainability (how easy it is to change)
- *Compare and contrast* literals and magic numbers, taking into consideration:
    - When is a magic number a literal?
    - When is a literal a magic number?
- Given a code snippet, *identify* which literals are magic numbers, and which are not
- *Define* constants. *Explain* their relationship with magic numbers
- *Define* the rules for naming constants

- *Compare and contrast* constants and variables

Strings

- Define concatenation
- Define slicing and substrings and explain their relationship
- Define escape sequences and enumerate the escape sequences for:
    - A tab character
    - A newline character
    - A single or double quote character
    - A backslash
- Define whitespace
- Implement code that:
    - Accesses individual characters in a string
    - Gets a copy of the string that is all lower case
    - Gets a copy of the string that is all upper case
    - Gets a substring that
        - Starts and ends in the middle
        - Starts at the beginning and ends somewhere in the middle
        - Starts somewhere in the middle and ends at the end
    - Remove whitespace at the front and end of a string using strip(), lstrip() and rstrip()
    - Split a string into a list of substrings, using `split()`
    - Join a list of strings into a single string, using `join()`

Functions

- Describe functions as the fourth type of program control
- Define code duplication and explain how it causes:
    - More bugs
    - Harder to maintain code
- Define the parts of a function and identify them in code:
    - Name
    - Parameters
    - Body
    - Definition
    - Arguments
    - Call
    - Return statement
- Explain what a return statement does in terms of a function and its caller
- Identify where functions can be legally called in code

- Identify problems that occur when a called function does not return a value when the calling code expects it to. i.e. Consider **`print(sum(x, y))`** but sum does not have a return statement
- Explain what receiving an error containing NoneType indicates
- Define `None` in terms of when it will appear while you debug your code
- Define scope and local variables. Describe what errors may occur if a function tries to access another function's local variables.

Program Design

- Define modularity and explain the benefits of a modular program
- Define helper functions
- Given a large problem, outline a program design that breaks down the problem into subproblems
- Define, compare and contrast top-down implementation and bottom-up implementation. (Hint: do so in terms of a program design outline)
- Advocate for testing as you go
- Define readability and explain its benefits
- Improve the readability of supplied code by applying the methods listed above
- Advocate for testing as you go
- Provide examples of good places to comment code
- Identify excessive comments in supplied code
- Advocate for testing as you go

2-dimensional Lists

- Define a 2 dimensional list
- Explain how to and/or implement:
    - Instantiate an empty 2d list
    - Instantiate Height x Width 2d list with a particular value (or set of values)
    - Access/Iterate over a row in a 2d list
    - Access/Iterate over a column in a 2d list
    - Access a single value in a 2d list
        - Which [] is the row number?
        - Which is the column number?
- Given a 2d list, access a particular value or set of values from the list
- Explain why a 1d list of strings can be considered a 2d list

Mutability

- Define mutable, and give examples of type(s) that are mutable
- Define immutable, and give examples of type(s) that are mutable
- Compare and contrast assignment of mutable and immutable variables

- Define a reference in terms of data and its location
- Define, compare and contrast deep and shallow copy
    - Remember that “deep copy” of a 2-d list is more complicated than “deep copy” of a 1-d list.
- Implement a deep copy in the following ways:
    - With a loop
    - With list()
    - With splicing
- Explain the impact of mutable parameters to a function in terms of shallow copy and scope

Dictionaries

- *Define* a dictionary in terms of association, ordering, keys and values
- *Explain* the constraint applied to keys with respect to mutability and uniqueness
- *Explain* why there are less constraints on values
- *Read, write and delete* entries from a dictionary
- *Define* and *utilize* the following dictionary methods:
    - `keys()`
    - `values()`
    - `get()`