# CS 111 Review

(This material is excerpted from a long Google Doc that was prepared for CS 111 students in the spring of 2022. My thanks to Eni Mustafaraj and the CS 111 staff for sharing it with me.)

## Midterm Exam

**Material for the 1st exam**

- Python expressions and statements
- Variables and assignments.
- Simple data types and operations on these: numbers (integer and floats), strings, booleans, `None`.
- Functions:
    - understanding the difference between function definition and function invocation.
    - function parameters
        - The name of parameters do not matter as long as they are used consistently.
        - In a function invocation frame, each parameter denotes a local variable initialized to the argument value.
    - understanding the difference between `return` and `print`.
    - The invocation frame model explains function calls.
- Scope:
    - the locality of parameters and other local variables assigned within a function body.
    - the `global` declaration for declaring a variable global within a function.
- Conditional statements: `if` statements with optional `elif` and `else` clauses.
- Sequences:
    - lists are mutable sequences of values. You can both change indexed slots and add and remove indexed slots from a list.
    - tuples are immutable sequences of values.
    - strings are immutable sequences of characters.
- Iteration:
    - Iterations are repeated updates to state variables, as expressed in iteration tables via iteration rules
    - Iterations are expressed in Python using loops:
        - `while` loops
        - `for` loops range over lists and are just `while` loops in disguise
        - loop gotchas:
            - premature return from the function (or early return)
            - updates to state variables in wrong order
        - Sometimes you **want** to return early from a loop via `return` or `break`
        - nested loops

    - Iterations can also be expressed with list comprehensions, which are a compact notation for mapping, filter, and appending patterns.
- Understanding how to use functions and objects from their contracts.
    - The `turtles` library has lots of methods with contracts that you've used. Similarly, turtleBeads.
    - you've also seen contracts for operations on sequences (lists, tuples, strings).
- Abstraction:
    - Python functions abstract over behavior.
- Problem solving strategies:
    - Divide/conquer/glue (or Divide/Solve/Combine)
    - Designing iterations (loops) with iteration tables and iteration rules
    - Incremental programming: work towards a full solution by starting with a simple solution that does very little adding more features and detail
- Memory Diagrams
    - Memory Diagrams show the state of a program involving variables and values, including mutable values like lists and objects.
    - They are especially important for understanding:
        - assignment: changing the value stored in a variable, list slot, or object instance variable.
        - adding or removing slots in a list.
        - aliasing: the same mutable value can be accessed by multiple paths.
        - == vs. **is** for testing value equality.
    - Be able to translate from a given memory diagram to python code that produces such a diagram and also, given python code, draw the corresponding memory diagram to reflect the state of the variables and values.

# Final Exam

Topics covered on the final exam include the following:

- Keyword arguments for sorted: Python allows functions to be called via key when using `sorted`. For example, `sorted(seq, key=cityState, reverse=True)` where `cityState` is a function that returns a tuple with the city and the state extracted from a given item in `seq`.
- Nested loops
- Debugging
    - Test cases
    - Tracing
    - Counterexamples
- Recursion
    - Base case and recursive case

    - Wishful thinking: when defining a recursive function, **assume** it works correctly on all smaller problems. If you define it correctly based on this assumption, it **will** work correctly on all smaller problems.
    - Explained by the function frame invocation model
    - Graphical examples involving turtles; music examples involving drums
    - Invariance: state that is unchanged by recursion (e.g. position and heading of turtle)
    - Fruitful recursion: when recursive function returns a value.
        - Classic numerical examples: factorial and fibonacci
        - Can process lists recursively as well as iteratively
    - Parameters name variables that hold the actual arguments
    - Local variables are also shown as boxes inside the frame.
    - The body of the function is evaluated relative to the local and global variables.
    - If the body assigns to a global variable, that variable must be declared global inside the function
- Multiple assignment (assignment with tuples on the left hand side of =). For example: `a, b = (4, 8)`.
- List comprehension and operations such as mapping and filtering.
- Dictionaries:
    - Map keys to values.
    - Use subscripting notation, [...], to get and set value for key.
    - `.keys()`, `.values()`, `.items()`, `.get()` methods are helpful
    - common use is to keep the frequency count of items.

- Advanced combination of these mutable data structures: dictionaries of dictionaries, list of dictionaries, dictionaries of lists. Conversion from one data structure to another.
- Files are persistent storage.
    - Handling various files types (text, csv, json)
    - `csv.DictReader, csv.DictWriter`
    - `json.dump` and `json.load`
    - Operations for reading and writing files include `open`, `.readlines`, `.readline`, `.read`, and `.write`.
        - The special `with … as` statement for file manipulation will implicitly close files upon completion.

    - Common operations on files and directories include `os.getcwd`, `os.listdir`, `os.path.isfile`, `os.path.isdir`, `os.path.exists`, and `os.path.join`.
        - In conjunction with recursion, these operations can be use to visit every file and directory in a tree rooted at a directory.
        - `os.listdir` returns a list of file names without any directory prefix information. It is often necessary to use `os.path.join` or string concatenation to create appropriate relative or absolute filenames for other operations.
- Exceptions:

- used for exceptional situations -- e.g., opening a file that does not exist, dividing by zero.
- handled by `try`/`except`.