CIS 130 Exam #1 Review Suggestions

* last modified: 3-21-07, 2:45 pm

* remember: YOU ARE RESPONSIBLE for course reading, lectures/labs, and especially anything that's been on a homework, in-lecture exercise, or lab exercise; BUT, here's a quick overview of especially important material.

  (and, yes, the not-really-a-week-8 lab exercise is fair game, too; see its posted example solution on the course Moodle site.)

* The test covers through and including sentinel-controlled loops.

  (Note that lists/arrays and file input/output will NOT be on this exam. They will be fair game for the final exam!!)

  The test will particularly focus on the material covered since Exam #1. (The Exam #1 material still applies --- we have built atop it, so it cannot be avoided! --- but the focus of most questions will be the "new" material.)

* you are permitted to bring into the exam a single piece of paper (8.5" by 11") on which you have handwritten whatever you wish. This paper must include your name, it must be handwritten by you, and it will **not** be returned.

  Other than this piece of paper, the exam is closed-note, closed-book, and closed-computer.

* this will be a pencil-and-paper exam, but you will be reading and writing Python code, statements, and expressions in this format.

  * note that you could be asked to write Python expressions and functions;

    (note, too, that answers may lose points if they show a lack of precision in terminology; for example, if I ask for a literal and you give a whole expression, instead)

  * note that I could ask you what given Python code does or means; I could give you one or more expressions, a function, etc., and ask you what it does or what would then be shown in the python interpreter in a given situation (or how you could write a call to use it, etc.)

  * you could be asked to modify a piece of code or function or module, or to correct a segment of code or a function or a module, as well;

* note: you are still expected to be comfortable with and use the design recipe; you are still expected to be familiar with and use the course style standards (both those you needed to know for the first exam, and those that have been added during this time period).

* (note that there could be questions about the class style standards, and your grade for test questions may be affected by whether you follow those class style standards in your answers.)

* (note that there could be questions where you are given code and/or design recipe elements, and asked if they are "legal", if they meet class coding standards, if they are syntactically or logically correct; you might have to correct code or design recipe elements that are "incorrect" or "illegal" in some sense.)

* the **bool** type

    * what are the only two **bool** literals?

    * what kinds of expressions return values of type **bool**?

    * you should be able to write a function that takes a **bool** parameter; you should be able to write a function that returns a value of type **bool**. You should be able to set a variable's value to be the result of a conditional/Boolean expression.

* the **string** type (**str**)

    * how do you write a string literal in Python? (Note that there is more than one way! We covered two different ways.)

    * What does the + operator do/mean when it is between two strings?

    * You should be able to set a variable's value to be a string result.

* **conditional** expressions: **relational** expressions, **boolean/logical** expressions

    * how do you write conditional/Boolean expressions in Python?
        * what are the relational operators provided by Python?

        * what are the Boolean operators provided by Python?

        * an expression that involves relational and/or Boolean operators is a conditional/Boolean expression; (note that such expressions may involve arithmetic sub-expressions, too --- e.g., seeing if the sum of two values is less than another...)

        * what is the type of such an expression? (not **int**, but...)

        * given a description of such a desired operation, you should be able to write an appropriate Python expression implementing that operation.

        * Need to be comfortable reading, writing, debugging conditional/Boolean expressions.

    * (be careful of the *difference* between = and ==) !!

    * Beware of common pitfalls (e.g., know how to properly see if a value is between two other values)

* **numeric intervals**

    * should be comfortable reading and writing numeric intervals as discussed in lecture and in the assigned reading;

    * given a marked number line, write the equivalent interval; given an interval, be able to mark a number line so that it corresponds to that interval.

    * be able to write a Python function that determines whether an argument is within a specified interval or not;

    * need to be able to write the interval corresponding to a Python expression.

* **function that test conditions**

    * Should be comfortable reading, writing, and calling boolean functions (functions which return a value whose type is **bool**.)

* **branch structure / Python if-statement**

    * what are the 4 basic structures of programming? (sequence, branch, procedure/function, and repetition)

    * what is one of the statements that Python provides for implementing the branch structure?

    * what is the syntax of the Python if-statement? What are its semantics?

    * should be very comfortable with the Python-expected indentation for if-statements.

    * be comfortable with the if-elif-else "style"/pattern of using particularly-nested

Python if-statements to express a situation where you wish to make exactly one choice from a collection of three or more choices;

- be comfortable designing, reading, and writing if-statements; you should be able to read an if-statement, and tell what it is doing; you should be able to given its output.

- be comfortable designing, reading, and writing **conditional functions**; given a conditional function, you should be able to read, write calls to that function; you should be able to indicate what such a function call would return as its value.

- what (according to our class style standards) are the minimum examples/tests that should be included for a **conditional function**?

- should be able to read a flow chart depicting a branching situation;

* **local variables**

  - how is a local variable declared? Where is a local variable declared? Once declared, in what part of a program is that local variable "visible"? (This is called its **scope** --- the part(s) of a program where a particular name has meaning.)

  - how can you set a local variable to a value? (You should know at least **three** ways...)

  - remember that a local variable is another Python identifier --- so, what are the course style standards about how it should be named?

* **assignment statements**

  - what is an **assignment statement**? What does it do? How do you write one? When should you use one?

  - what do we mean by the LHS and RHS of an assignment statement? what can be on the LHS of an assignment? what can be on the RHS of one?

  - be comfortable reading and writing assignment statements; I am quite likely to see if, given a fragment including assignment statements, whether you can say what the values of various variables would be at the end of that fragment.

  - be able to write an assignment statement that **increments** a particular local variable; be able to write an assignment statements that sets a particular local variable to some expression.

  * course style standard to keep in mind: given the kind of parameters that we are using currently, note that it is poor style to *change* a parameter within a function. (Now that we have the **raw_input** function and **assignment** statements, you now might be tempted to do this --- but remember that it is poor style, and AVOID doing so.)

    (if you want to change a variable within a function, change a local variable instead --- for example, assign a parameter value to a local variable, and then change that local variable as desired instead of the parameter itself.)

* **testing functions**

  * what do we mean by a "testing" function? Should be comfortable reading/writing/debugging a function whose purpose is to test another function's examples.

  * how can you write a function that implements an interactive interface to a non-interactive function (particularly to a "pure" function)? Should be able to read/write/debug such a function.

* **interactive input/output**

  * you should be comfortable reading/writing string literals in Python

  * should be quite comfortable with how **raw_input** and **print** work;

  * how can you get spaces in your output if you want them? how can you avoid print ending with a newline?

  * you should be able to write well-designed raw_input calls to prompt a user for interactive input (including numeric input)

* **void functions** and **function with no parameters**

  * in a contract, what is the return type for a function that doesn't return anything? If it doesn't do anything, what are some examples of what such functions might do instead?

  * what statement does not need to be included within a void function?

  * how do you call a void function?

  * how do you write a contract for a function that doesn't return anything? How do you write its header? How do you call such a function?

- how do you write a contract for a function that doesn't expect any arguments? How do you write its header? How do you call such a function?

* **loops/repetition**
    * remember, repetition is one of the four basic structures of programming (sequence, branch, procedure/function, and repetition)
    * know that a **while statement** is one of several statements that Python provides for implementing the repetition structure;
    * what is the syntax of the Python **while** statement? what is its semantics?
    * what do we mean by a loop condition? by a loop body?
    * you need to be familiar with the two classic "patterns" we discussed in connection with loops: count-controlled, and sentinel-controlled.
        * you should be able to read, write, and correct loops using the classic versions of each of these;
        * (you should know that these are simply two very common patterns --- they are not your ONLY possibilities! 8-) You should be able to read/write more general **event-controlled** loops, too, as well as other loop variants.)
        * yes, you do have to know how to write/read/correct a classic count-controlled loop that happens to use a while loop;
    * be comfortable reading and writing while loops; you should be able to read a while loop, and tell what it is doing; you should be able to give its output; you should be able to tell how many times its body is executed.
    * what is an infinite loop? how can it be avoided?
    * what is some other classic loop concerns? (does it go one too many, or one too few, times? is it ever entered at all?)
    * should be able to read a flow chart depicting a looping situation;