CSSE 120 – Introduction to Software Development

Exam 1 – Format, Concepts, What you should be able to do, and Sample Problems

Format: The exam will have two sections:

- **Part 1: Paper-and-Pencil**
  - **External resources allowed**: An 8.5 by 11 sheet of paper (back and front) "cheat sheet" with whatever you want on it (printed or handwritten).
    - You would be wise to create your own cheat sheet (working with someone else is fine) as that will maximize both your learning and your score on the exam.
  - **Points**: Approximately 25 of 100

- **Part 2: On-the-computer**
  - **External resources allowed**:
    - Any printed or handwritten materials you choose to bring, including notes, books, handouts, printouts, whatever
    - Your own computer and any external drives attached to it
    - Your own SVN repository
    - Anything on the Internet, subject to: ***During the exam, you may not communicate in any way with any human being other than your instructor (or other exam proctor). No email, no chat, etc.***
  - **Points**: Approximately 75 of 100

In the following, we have tried to list ***everything* that you might be expected to demonstrate on the exam**, plus ***sample problems*** for as much of that as practical. But please note:

- This is not a *contract*; it is only our *best-effort* to list everything you might be expected to demonstrate on this exam.
- The sample problems that will follow are just that – *samples*. **The problems on the exam may look both *similar to* and somewhat *different from* the samples.**

Concepts of software development that you might be asked to *explain* include:

1. The difference between a *specification* and an *implementation*, and what a *specification* of a function should include. What a *pre-condition* is and why they are important.

Critical

2. *Documentation*: how and why we put *internal comments* and *documentation strings* (*doc-comments*) in our programs
3. *Software development tools*: what is provided by a typical, modern:
   a. Integrated Development Environment (IDE)
   b. Version control system
   c. βββ Debugger

Items with a triple beta βββ next to them, like this one, are important items that you will NOT see on Exam 1

4. *Software development processes*:
   a. βββ What are some important *phases* of software development?
   b. What is *procedural decomposition*?

   Critical

   c. What is a *compiler*? An *interpreter*?
   d. What is the difference between a *compile-time error* and a *run-time error*? *Syntax* and *semantics*?
   e. What is an *algorithm*?

This entire section (on functions) is critical

5. Key ideas of *functions*, including:
   a. Why are functions useful?
   b. What is the difference between a *function call* and a *function definition*?
   c. How does one send information *to* a function? *Back from* a function? What is a *parameter* and how is it used?
   d. What does it mean when we say that *variables* are *local* to a function? Why is the locality of variables a useful characteristic?
   e. Where does execution go when a *function call* is executed? When the function completes its execution? When a *return* statement is encountered?
   f. When a variable is used as an *argument* in a function call, does that send the *name* of the variable or the *value* of the variable to the function? How about when a variable is *returned* from a function?
6. Key ideas of *object-oriented programming*, including:
   a. What makes objects different from traditional data types, namely: ***objects know stuff*** (stored in *instance variables*) and ***can do stuff*** (via *methods*)
   b. βββ Why object-oriented programming is valuable
   c. The difference between a *function* and a *method*, and the different notations for invoking them
   d. The difference between an *object* and a *class* to which that objects belongs

   Critical

   e. βββ The difference between *accessor* and *mutator* methods
7. βββ What is the difference between the **`int`** and **`float`** data types? What are the limitations of each? When should use one and when the other?

Critical

8. What is a *sequence*? Why are they important? What does it mean to *index* into a sequence? What is the difference between the *sequence* types: *list*, *tuple*, *string*? For each, when should one use it instead of another sequence type?
9. βββ The implications of the fact that variables in Python are *names* that point to *values* in memory, that is, variables are *references* to their values. How to use *box-and-pointer diagrams* to trace, understand and depict the behavior of variables that reference their values. Exactly what an *assignment statement* does; what a *function call* does regarding *actual arguments* and *formal parameters*. What is a *mutator* and why is it useful? dangerous?

Concepts that you might see on *code* that you *read* and *write* include:

> ***Sample problems*** of each of these items appear ***later in this document***. If you don't understand what an item here is asking, see if the example problem clarifies matters for you.
>
> *Items with a triple beta βββ next to them are important items that you will NOT see on Exam 1.*

1. **Variables and assignment,** including βββ simultaneous assignment (`x, y = …, …`) and βββ operator assignment (`sum += …`)
2. **Data types**: *int*, *float*, sequences (*lists*, *strings*, *tuples*, *range* expressions)
3. **Arithmetic and character expressions**, including those involving:
   - Operators: `+ - * / // % **`
   - Math functions: `abs cos sin pi sqrt`
   - βββ Character functions: `ord chr`
4. The ***input*** function, including:
   - Providing a prompt
   - Converting an input string into a number (integer or floating-point) using ***int*** and ***float***
   - βββ Stripping whitespace from the beginning and end of an input string (using ***strip***)
   - βββ Splitting an input string into a list of strings (using ***split***) and then converting the strings in the list into appropriate types
5. The ***print*** function, including:
   - Printing on multiple lines or on the same line
   - βββ Using a string's ***format*** method and associated format specifiers to do formatted output, especially: columns lined up on decimal points, centering
6. **Sequences**: *Lists*, *strings*, *tuples* and *range* expressions. Including:
   - *Indexing* and βββ *slicing*, including negative indices. Accessing characters inside strings inside lists, etc.
   - The ***len*** function
   - Concatenation (`s1 + s2`) and βββ duplication (`s * n`)
   - βββ String methods like: ***capitalize count find format index join lower replace strip split title upper***
   - βββ List methods like: ***append count index insert remove reverse sort***
7. **Definite loops**, including:
   - *Counted* loops through a ***range*** expression
   - Looping *directly* through a list or string
   - Looping through a list or string using its *indices* as generated by a ***range*** expression
8. **Functions and methods**, including:
   - Function *definitions*, including parameters
   - Function and method *calls*, including those with actual arguments
   - *Returning* a value from a function and capturing/using returned values
   - βββ *Mutators* and mutable parameters
   - βββ *Optional parameters* – defining, using
   - Functions that call functions
9. **Objects**, including statements that:
   - *Construct* an object
   - Apply a *method* to an object
   - Reference an *instance variable* of an object

   zellegraphics and create as examples of *classes*, *constructors*, *methods* and *objects*
10. **Conditionals**, including:
    - The three forms:
      `if`
      `if-else`
      βββ `if-elif-elif...-else`
    - *Relational operators* on numbers/strings: `<  >  <=  >=  ==  !=`
    - *Boolean operators*: `and  or  not`
11. ***import* statements**, in two forms:
    ```
    import blah
    import blah as foo
    ```

For the *Paper-and-Pencil* portion of Exam 1, students should be able to:

1. **Trace by hand short snippets of code (less than 15 lines or so)** and show:
   - ***what gets printed,*** or the
   - ***values of indicated expressions***.

2. **Explain important concepts of software development,** chosen exclusively from the list on page 2.

Sample problems appear ***later in this document***. If you don't understand what an item here is asking, see if the example problem clarifies matters for

For the *On-the-Computer* portion of Exam 1, students should be able to:

1. ***Write short programs and/or functions*** that are examples of the ***input/compute/output pattern***. Be able to:

   a. Use the `input` function to get input from the console, including:
      - Provide a prompt
      - Convert an input string into a number (integer or floating-point) using the `int` and `float` functions

   b. Use *variables* to store the input and perform numeric computations using:
      - Operators: `+` `-` `*` `/` `//` `%` `**`
      - Functions: `abs` `cos` `sin` `pi` `sqrt` `round`

   c. Use *print* to display results on the console, all on one line or on separate lines

2. ***Define functions*** that have ***parameters*** and (possibly) ***return values***. Be able to:

   a. Write the `def` portion of a function definition, given (in ordinary English) the name of the function and a description of its parameters.

   b. Write the *function body*, using the *parameters* and other *local variables* as needed. Display an understanding of:
      - The fact that a parameter is a name for a *value* that comes into the function
      - The relationship of parameters and other local variables to variables with the same name outside the function
      - When and why to introduce local variables

   c. *Return* a value if called for by the problem

3. ***Call (invoke) functions,*** both ordinary functions and *methods*, and use the ***returned value*** (if any), perhaps by capturing it in a variable. This includes calling functions that you write and functions that you did *not* write (but use).

4. Use ***definite loops*** and ***sequences***

   a. Write a *counted loop*, that is, a loop that iterates a given number of times, by using a *range* statement, in any of its three forms: *range(n)* *range(m, n)* *range(m, n, d)*

   b. Use the *loop variable* as called for by the problem.

   c. *Iterate through sequence* in either of two ways, as necessary:

Looping *directly* through a sequence, e.g.

```
for thing in list_of_things:
    ... thing ...
```

Looping through a sequence *using its indices* as generated by a *range* expression, e.g.

```
for k in range(len(list_of_things)):
    ... list_of_things[k] ...
```

   d. Use the *Accumulator loop pattern* to accumulate things like a:

      - sum
      - count
      - product
      - list
      - string
      - coordinate or size of a graphics object
      - βββ max/min

   e. Use the *len* function to obtain the length of a sequence. *Index* correctly (starting at 0).

   f. Iterate through two equal-length sequences in parallel.

   g. Iterate through a sequence in a problem that requires the index variable to be used in more than one way at each iteration.

5. Use ***conditional*** statements, in any of their 3 forms:

   `if` `if-else` βββ `if-elif-elif...-else`

   - Use *comparison operators*: `==` `!=` `>` `<` `>=` `<=`
   - Use *Boolean operators*: `and` `or` `not`

6. Use the *wait-until-event loop pattern*, using a *while* statement and *break* expression.

7. Use ***objects***:

   a. *Construct* an object that is an instance of a *class*

   b. Apply *methods* to the object

   c. Reference *instance variables* of the object (but note: usually we use *accessor* methods instead of directly accessing the object's instance variables)

Also, display an understanding of:

   d. How to determine what methods apply to an object

   e. The distinction between an object and a class that it is an instance of

8. ***Apply*** the above to *zellegraphics*:
   a. Construct (and hence display) a GraphWin. Wait for the user to click the mouse.
   b. Construct and use a Point, Line, Circle, Rectangle, Polygon, Text, Entry
   c. Apply methods to the above, including (*not all of these apply to all of the above!*):
      - *draw*
      - *undraw*
      - *move*
      - *getMouse*
      - *close*
      - *getters* like:
        - *getX*
        - *getWidth*
        - *getY*
        - *getHeight*
        - *getCenter*
        - *getRadius*
      - *setters* like:
        - *setFill*
        - *setOutline*
   d. Do an animation (using *time.sleep*)

9. ***Debug*** your code:

   a. Use Eclipse to correct *compile-time errors* like this example:
   b. Use the *red error messages in the Console window* to know the line at which the program broke and the general nature (at least) of the error.
      Use the *blue link in the Console window* to see the line at which the code broke.
   c. Use either *print* statements or the *debugger* to track down harder-to-diagnose run-time errors

10. ***Test*** your code: Supply calls in *main* or elsewhere that call your functions with parameters that help test your functions, printing returned values as appropriate.

11. ***Document*** your code, using appropriate *documentation strings* (doc-comments) and *internal comments* (with # signs)

12. ***Submit*** your code, using SVN as usual.