CSSE 120 – Introduction to Software Development

Exam 1: *Format* and *What you should be able to do*

Table of Contents

Format of the exam

Honesty Pledge:

At the beginning of the exam, you will receive an Honesty Pledge which you should have read before the exam – it is available from the above link, or visit the CSSE 120 web site, under *Resources*, then *Materials to help you prepare for CSSE 120 exams*. You will sign and turn in that pledge at the end of the exam.

Two parts:

- **Part 1:** ***Paper-and-Pencil.***

  For this part, the ONLY external resource you may use is a single 8½ by 11 sheet of paper, with whatever you want on it, typed or handwritten or a combination of the two. You may use BOTH sides of the sheet. You must have prepared the sheet *before* beginning the exam.

  - It is best if you create your own sheet (working with someone else is fine) as that will probably maximize both your learning and your score on the exam.

- **Part 2:** ***On-the-computer.***

  For this part, the only external resources allowed are:

  - Any written material you choose to bring to the exam: books, handouts, notes, etc.
  - Your computer and anything on it.
  - Your own SVN repository
  - Anything directly reachable from the CSSE 120 web site.

  You may *not* use any search engine (like Google).
  (Exception: If a search engine is embedded into a site directly reachable from the CSSE 120 web site, and is restricted to that site, then you may use that limited search engine.)

Communication:

For both parts of the exam, ***you must not communicate with anyone*** except your instructor and his delegates, if any.

Time limit:

You have ***two hours*** to complete the entire exam – its *paper part* and its *computer part*. You will receive both parts at the beginning of the exam. You must complete the paper part (using only your prepared 1-page-front-and-back sheet) and turn it in before you begin work on the computer part.

Essentials

(if you can't do these, you probably will not pass the exam!)

Be SURE you can:

1. ***Implement*** and ***test*** a function, per its ***specification***
2. Solve ***input-compute-output*** problems.
3. ***Use parameters*** and ***return*** values.
4. In ***zellegraphics***: Construct a window. Construct and draw objects (circles, rectangles, etc) on it. Have the window wait for a mouse click before closing.
5. Succeed in simple ***summing*** and ***counting*** problems.
6. Know ***what to do*** when a ***compile-time error*** occurs, and when a ***run-time error*** occurs.

Example: **Session 2**, ***m2_input_compute_output***

Example: ***digits_in_power*** function in **Session 4**, ***m4_calling_functions_returning_values***

Example: **Session 3, *m2_using_objects***

Example: **Session 6, *m2_summing_and_counting***

Read the ***green*** specification.

Do NOT change the number, order or meaning of the ***parameters***, nor the function name.

If the specification says to ***return*** something, then `return` it and do NOT print it (in the function).

Test the function in a testing function, ***by calling the function*** with appropriate parameters. Provide as many tests (calls) as the problem requires. It is OK (and normal) to *print* in the *testing* function.

What you should be able to do – Summary

This is not a *contract*; it is only our *best-effort* to list everything you might be expected to demonstrate on this exam.

See the pages that follow for an elaboration of this summary.

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

1. ***Read*** short snippets of code.
2. ***Write*** short snippets of code.
3. ***Explain*** important concepts of software development, chosen from a short list (below)

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

7. ***Write short programs and/or functions*** that are examples of the ***input-compute-output pattern***.
8. ***Call (invoke) functions*** and ***methods.***
9. ***Implement and test functions*** that have ***parameters*** and (possibly) ***return*** values, per the function's ***specification***.
10. ***Use objects: construct*** instances, use ***methods***, reference ***fields***, and apply all this to ***zellegraphics***.
11. Use ***counted loops,*** that is, `for ... in range(...)` ***statements,*** especially as applied via the ***Accumulator Pattern*** (***summing***, ***counting***, ***in graphics***, etc).
12. Use ***conditional*** statements, with ***relational operators*** and ***Boolean operators***.
13. ***Debug***, ***test***, and ***submit*** your code.
14. ***Apply*** the *Concepts* below as needed to accomplish the above.

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

1. ***Variables*** and ***assignment***
2. ***Data types***: `int` `float` `string` `bool`

   The `type` function to tell the type of an object.
3. ***Arithmetic operators*** and ***expressions*** and functions from the `math` module.
4. The `input` and `print` functions.
5. ***Functions*** and ***methods***, including ***calling*** (***invoking***) and ***defining*** them; ***parameters*** and ***arguments***; ***returned values***; and functions that call functions.
6. ***Counted loops***, i.e., loops through a `range` expression
7. ***Objects***, including ***constructors***, ***methods*** and ***fields***.
8. ***Conditionals***, ***relational operators*** and ***Boolean operators***.
9. The ***flow of execution***: sequential, and per function calls and returns, conditional statements and loops. The roles of `main` and `import` statements.
10. The ***scope*** of variables, per ***namespaces***.
11. ***zellegraphics*** as an example of using objects and as a rich place in which to apply all the above.

What you should be able to do – Elaboration

See Page 3 for a summary of this elaboration.

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

1. ***Read*** short snippets of code.
   a. ***Trace*** short snippets of code (less than, say, 10 lines or so) and show ***what gets printed*** or the ***values of indicated variables/expressions***. Especially:
      - Following the sequence of execution through:
        - Function calls (including functions that call functions)
        - FOR loops
        - Conditionals
      - Sending arguments to functions and capturing returned values (noticing the scope of variables).
      - The effect of accumulator statements like `x = x + 1`.

   > See later in this document for a list of ***concepts*** that you might see on code you ***read*** and code you ***write***.

   b. ***Indicate errors*** in short snippets of code:
      - ***Syntax*** errors: something wrong or missing in *notation*
      - ***Semantic*** errors: does *such-and-such*, should do *so-and-so*

2. ***Write*** short snippets of code, especially:
   a. `range` expressions for common ranges
   b. ***Counted loops*** (that is, `for ... in range(...)`) that generate/print simple sequences
   c. ***Function definitions***, including those that have ***parameters*** and/or ***return values***
   d. ***Function calls***, including follow-up code that uses a returned value
   e. ***Conditionals*** with ***relational*** and ***Boolean operators***

3. ***Explain*** important ***concepts of software development***, chosen from:
   a. The difference between ***syntax*** and ***semantic*** errors.
   b. The difference between a ***specification*** and an ***implementation***, and what a ***specification*** of a function should include.
   c. ***Why functions are useful*** and important
   d. ***Documentation:*** how and why we put ***internal comments*** and ***documentation strings*** in our programs.
   e. ***Software development tools:*** what is provided by a typical, modern:
      - Integrated Development Environment (IDE)
      - Version control system
      - Debugger

   > While you might see some problems of type #3, don't expect a lot of such questions and don't expect them to be deep. A simple understanding of these concepts is adequate.

   f. Key ideas of ***object-oriented programming***, in particular:
      - What makes objects different from traditional data types? Answer: objects ***know stuff*** (stored in ***fields***) and can ***do stuff*** (via ***methods***)
      - The difference between a ***function*** and a ***method***, and the different notations for invoking them
      - The difference between an ***object*** and a ***class*** to which that objects belongs
   g. The difference between the `int` and `float` data types. The limitations of each; which you should choose when.
   h. What ***pair programming*** is, and why it is useful

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 `input` to get input from the console, including:
      - Provide a ***prompt***
      - Convert an input string into a number (integer or floating-point) using `int` and `float`
   b. Use ***variables*** to store the input and perform numeric computations using:
      - Operators: `+` `-` `*` `/` `//` `%` `**`
      - Functions and constants from the `builtins` and `math` modules, including:
        `cos` `sin` `sqrt` `pi` `abs` `round`
        and others that you should be able to look up with the "dot trick"
   c. Use `print` to display results on the console, with or without appropriate strings that explain the results
2. ***Call (invoke) functions*** and ***methods***
   a. Whether ***built-in***, defined in the ***current module***, or from an ***imported module***.
   b. Use the ***returned value*** (if any), perhaps by capturing it in a variable.
3. ***Implement and test functions*** that have ***parameters*** and (possibly) ***return*** values, per the function's ***specification***.

   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. Implement the ***function body***, using the ***parameters*** and other ***local variables*** as needed, per the function's ***specification***. Display an understanding of:
      - A ***parameter*** is a name for a value that comes *into* the function *from the caller*.
      - The function can ***return*** a value *to the caller* with a `return` statement.
      - ***Scope*** and ***namespaces***: parameters and other local variables have no direct relationship to variables with the same names in other functions.
      - ***Coding to a specification:***
        - You may NOT change the number, order or meaning of the parameters, nor the function name.
        - Your implementation must meet the specification of its ***documentation string*** (displayed in **green** between the function header and body).
        - In particular, the function must ***return*** a value if called for by the specification and must ***not*** *print* anything unless the specification says to do so.
   c. ***Test the function*** in a ***testing function***, by ***calling*** the function with appropriate parameters.
      - ***If the tested function returns a value, print the returned value*** and compare it to the expected (correct) value for that test case.
      - Each such function call forms *one* test case. You should be able to use test cases that we supply as well as develop reasonable test cases on your own.
4. ***Use objects: construct*** instances, use ***methods***, reference ***fields***, and apply all this to ***zellegraphics***.
   a. ***Construct*** an object that is an ***instance*** of a ***class***
   b. Apply ***methods*** to the object

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

d. Determine what methods apply to an object and what fields it has, by using the ***"dot trick"***

e. Use the ***pop-up information that the "dot trick" displays*** to make reasonable guesses for what arguments are needed in constructing an object or applying a method.

- Be able to use the dot trick even when the variable of interest is a parameter (and hence its type is not known to the dot trick).
- Be aware of the special role of the `__init__` method for constructors and how to use it.

f. Understand the distinction between an ***object*** and a ***class*** that it is an ***instance*** of.

5. Use ***counted loops***, that is, `for ... in range(...)` ***statements***, especially as applied via the ***Accumulator Pattern*** (***summing***, ***counting***, ***in graphics***, etc).

   a. Use 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. Use the ***Accumulator Pattern*** in forms that include:

      • ***summing*** • ***counting*** • ***averaging***

      • ***products*** (including ***factorial***) • ***in graphical patterns***

6. Use ***conditional*** statements, with ***relational operators*** and ***Boolean operators***.

   a. ***Conditionals:*** `if` `if-else` `if-elif-...-else` Know when to use which of the above.

   b. Use ***relational operators:*** `>` `<` `>=` `<=` `!=` and especially carefully `==`

   c. Use ***Boolean operators:*** `and` `or` `not`

7. ***Debug***, ***test***, and ***submit*** your code.

   a. Use Eclipse to correct ***syntax errors*** like this example:

   ```
   45    oops - 1 = oops
   ```

   b. Use the ***red error messages in the Console window*** and the ***associated blue links*** to know the line at which the program broke and the general nature (at least) of the error

   c. Use the ***Debugger*** to track down harder-to-diagnose run-time errors

   d. ***Test*** your code: Supply calls (typically in testing functions) that call your functions with parameters that help test them, printing returned values as appropriate.

   e. ***Submit*** your code, using SVN Commit as usual.

8. ***Apply*** the above to ***zellegraphics:***

   a. Construct (and hence display) a `GraphWin`, and use `closeOnMouseClick` and/or `getMouse` to keep the window from disappearing prematurely.

   b. Construct and use a `Point`, `Line`, `Circle`, `Rectangle`, `Oval`, `Polygon`, `Text`, `Image`, `Entry`, or even (using the "dot trick") something similar that we add to zellegraphics just for the exam.

   c. Apply methods to the above, including but not limited to (*not all of these apply to all of the above!*):

      `draw` `undraw` `move` `closeOnMouseClick` `getMouse`

      getters like: `getX` `getY` `getP1` `getP2` `getFill` `getWidth` `getCenter` `getRadius`

      setters like: `setFill` `setOutline`

   d. Do an ***animation*** (using `time.sleep` and `move`)

9. ***Apply*** the ***Concepts*** below as needed to accomplish the above.

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

1. ***Variables*** and ***assignment***
2. ***Data types***: `int` `float` `string` `bool`

   The `type` function to tell the type of an object.
3. ***Arithmetic operators*** and ***expressions*** and functions from the `math` module, including:
   a. Operators: `+` `-` `*` `/` `//` `%` `**`
   b. `math` functions/objects: `abs` `cos` `sin` `pi` `sqrt`
   c. `builtins` functions: `min` `max` `round`
4. The `input` and `print` functions.
   a. Providing a ***prompt*** for input
   b. Converting an input ***string*** into a ***number*** (integer or floating-point) using `int` and `float`
   c. Printing ***string literals*** and values of ***variables*** together in sensible ways
5. ***Functions*** and ***methods***, including:
   a. Function ***definitions***, including ***parameters***
   b. Function and method ***calls*** (aka ***invoking*** them), including those with ***actual arguments***
   c. ***Returning*** a value from a function and capturing/using returned values
   d. Functions that call functions
6. ***Counted loops***, i.e., loops through a `range` expression in any of its three forms:

   `range(n)` `range(m, n)` `range(m, n, d)`
7. ***Objects***, including statements that:
   a. ***Construct*** an object
   b. Apply a ***method*** to an object
   c. Reference a ***field*** (aka ***instance variable***) of an object
8. ***Conditionals***, ***relational operators*** and ***Boolean operators***:

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

   `>` `<` `>=` `<=` `!=` `==`

   `and` `or` `not`
9. The ***flow of execution***: sequential, and per function calls and returns, conditional statements and loops. The roles of `main` and `import` statements.
10. The ***scope*** of variables, per ***namespaces***.
11. ***zellegraphics*** as an example of using objects and as a rich place in which to apply all the above.

Sections of the textbook that you read, all of which are relevant to the above:

- Chapter 2, sections 2.1 through 2.5
- Chapter 3, sections 3.1 through 3.7
- Chapter 5: sections 5.1 through 5.5, and 5.8