CIS 480 Final Examination Review Suggestions

* last modified: 12-07-05

* 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.

* 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, and you are expected to work individually.

  (Studying beforehand in groups is an excellent idea, however.)

* final is CUMULATIVE!
  * if it was fair game for exams #1 or #2, it is fair game for the final;

  * Thus, using the posted review suggestions for exams #1 and #2 in your studying for the final would be a good idea. (Note that they are still available from the public course web page, under "Homeworks and Handouts".)

  * studying your Exam #1 and Exam #2 would also be wise.

  * there may indeed be similar styles of questions on the final as on those exams.

* high points from the material SINCE Exam #2:

* **Focus on Modules**

  * now that you HAVE had a homework over this material, it is of course even more fair game...

  * **hiding names from from***

    * what kinds of names are **not** copied out when one imports using a **from *** statement?

    * this is not TRUE data hiding, however; why not? What is its real intent, then?

    * how can one achieve a similar effect by using **__all__**? In what order does Python handle these particular features when a **from *** is done?

  * **__name__** and **__main__**

    * what is the module built-in attribute **__name__** set to when a module is run as a top-level program file? what is it set to when the module file is imported?

    * how can this be used to write a module that can be both imported smoothly and run as a stand-alone program?

    * be able to tell, looking at a module, how it will behave when it is imported within something, and how it will behave when it is run as a top-level program file with **python**; be able to write a module that does particular things (that may differ) when run as top-level program file as opposed to when imported within something.

    * what's an example of something that one might have a module do differently when it is run as a stand-alone program?

* **Belated handy-dandy Python tools**

    * **dir**

        * built-in function, returns a list of all of the attributes (methods, simple data items, etc.) of an object;

        * be able to call this appropriately; recognize when it might be useful;

    * **system-defined names**

        * how can a system-defined name be recognized? Can/should a user define additional names using this convention?

    * **the __doc__ attribute**

        * what is a **docstring**? How (syntactically) is it coded? What kinds of things can have them?

        * what is the **__doc__** attribute? How is it set?

        * which Python built-in function uses **PyDoc**, a standard tool that knows how to extract and format docstrings into various nicely arranged reports, to generate a UNIX-man-page-like text report? Be able to call this appropriately; recognize when it might be useful.

* **Intro to Object-Oriented Programming (OOP) in Python**

    * you should know that an object includes BOTH data attributes and method attributes; you should know that in Python you can define a **class** and then create instances of that class, called **objects**.

    * how do you define a class in Python? how do you define data attributes and methods within a class?

    * how can you define a class so that it is a **subclass** of another class?

    * what is special about the **__init__** method of a class? What kind of a method is it? What is its purpose? How (using what syntax) is it typically invoked?

* given a class, you should be able to create instances of that class; you should be able to use and set its data attributes, you should be able to invoke the method attributes for that new instance; you should be able to write a subclass of that class.

    * you should be able to give the values for various expressions --- references to data attributes, method calls, etc. --- given the above information, also;

* you should be comfortable with how **inheritance** works in Python;

    * for example, given several classes (some of which are subclasses of each other) and some instances of those classes (possibly depicted as Python code, or possibly depicted graphically), you should be able to determine which version of what is being "used" in various expressions.

    * what is meant by **multiple inheritance**? How do you get it, in Python? In what "order" are ancestors searched when it is done?

* how is a method within a class written? how is it then invoked by a class instance/object? (Here, I am talking about methods that are intended to be methods for each individual class instance created...)

* within a class's methods, how do you refer to the data attributes of the calling instance? (Another way to look at this: within a class, how do you specify the difference between class-wide attributes and specific instance attributes?)

* what system-defined method name returns a string representation of an instance? You should know when you will see the results of this method for an instance (when is it called automatically?), you should be able to write this method for a class, you should be able to give the results of this method being invoked.

    * note: you are not responsible for **__str__** for this exam.

* **Intro to Exceptions in Python**

    * we discussed how exceptions can be used for more than error handling --- you should know some of the other things that they can be used for, too (for example, see p. 394 of Learning Python)

    * what does it mean to raise/throw an exception? what does it mean to catch an exception?

    * if your code does not catch an exception that is thrown, what gets invoked if it makes it all the way up to the top-level of the program? What is its usual action in this case? What is included in its response?

    * what kind of statement do you "wrap" around a statement that might cause an exception to be raised, if you would like to catch it?

* what kind of statement can you then use to catch a raised exception? How can you indicate that you are only catching a particular kind of exception? How can you indicate that you'd like to catch any exception?

    * how can you find out the type and value of a raised exception, once it has been raised?

* what is special about the **try-finally** combination? How is it used? Why would one use it? What are the restrictions on its use?

* in a **try-else:** combination, when is an **else**: clause action's done?

* be comfortable with **except:**, **except** *name*, **except** *name, data*, **except (***name1, name2***)**;

* how can you raise/throw an exception (with what Python statement?)

* with what Python statement can you indicate a condition that should be true, such that, if it is false, a certain kind of exception will be raised? What exception is this?

* given code containing the above, you should be able to tell what it would do; you could also be asked to write code involving the above, and you might be simply asked questions about the above, also.

* **Intro to GUI's and Python**

    * one possible question I MIGHT ask: be able to name at least one module/package available to help in writing GUI's using Python.

    * that's about it for this topic, in terms of the final --- do please note the links in the References section on the course web page, if you are interested.