## CSCI A201/A597 - Introduction to Programming I

## Fall 2017

# Programming Assignment 13

[100 points total]

## Due Date

- at 4:00PM, on Monday November 27, 2017 ← (right after Thanksgiving Break week) ←

## Work Policy

Programming Assignment 13 is to be done individually - no group solutions or cooperative work.
You may discuss the programming assignment with other A201/A597 students, but you have to write the solutions and implement programming answers by yourself. If you discuss the programming assignment with other A201/A597 students, you must acknowledge them by including their names in a comment at the top of the Python files you turn in for this programming assignment. Do not share any source code or programming assignment answers with any students.

### Notes

Keep a written log of your work for Programming Assignment 13, in a separate text document named **`pa13-log.txt`**, in which you annotate your work on these tasks.

At the top of the **`pa13-log.txt`** file, write:

- "A201/A597 Programming Assignment 13"
- your first and last name
- your IU username (that's the same as your IU email, *not* your numerical ID)

- in your `pa13-log.txt` file, write down notes about how you solved Programming Assignment 13 programming tasks.
  In particular, point out any difficulties, parts that may have been harder-than-usual to solve (as well as easier-than-usual), &c. Also (optionally) please include anything that you may have a question about, as well as any particularly good code you included in your solution, or anything else you may consider worth mentioning about it.

Solving A201/A597 programming assignments requires time - if you wait until the day when this assignment is due to start working on it, you may run out of time... we recommend that you start working on this assignment right away!

## Date Date

- At 4:00PM, before your next A201/A597 lecture section begins (either Monday or Tuesday, depending on your enrollment section).

## Work Policy

Programming Assignment 13 is to be done individually - no group solutions or cooperative work. You may discuss the programming assignment with other A201/A597 students, but you have to write the solutions and implement programming answers by yourself. If you discuss the programming assignment with other A201/A597 students, you must acknowledge them by including their names in a comment at the top of the Python files you turn in for this programming assignment. Do not share any source code or programming assignment answers with any students.

### Getting Help

- attend **labs** and **office hours**
- ask the AIs, UIs, and Instructor:
  the recommended way to contact A201/A597 instructors is with a **Canvas message**: this way, you'll reach all A201/A597 AIs, UIs and Instructor at once.
  Log in to IU Canvas, select "Inbox" in the left-side window toolbar, then select the "Compose a new message" icon at the top. Select "Course:CSCI A201...Fall 2017" → "To: Teachers" → "All in Teachers", and include A201 (or A597) in the Subject.
- consult the textbook
- consult lecture notes
- consult the references provided in reading assignments

## Tasks

This week's Programming Assignment tasks are about working with *files* and handling *exceptions*. Please review this week's Reading Assignment 13 material.

For Programming Assignment 13, it is *required* to include properly formatted docstrings with purpose statements and signatures for ***all*** functions you are asked to define. If you don't include properly formatted docstrings with purpose statements and signatures for all functions you define for Programming Assignment 13, your assignment will be *rejected*.

*Note:* save your IDLE interactive shell transcript into a file named `pa13-transcript.py` for your interactions with the Python scripts you write for all Tasks.

A. [30 points total]
   Task A consists of warm-up exercises for the `try...except` statement. Write a class named `Exceptional` and instantiate it once in the *main program* part of the script, then call each one of the methods that you write for solving Task A problems. Save your script in a file named `pa13-exceptional.py`.

   1. [10 points]
      consider the following Python code snippet:

```
try:
        num = int(input ("Enter an integer:"))
except ValueError:
        print ("Value Error")
else:
        print ("Integer entered.")
```

      Place the code above in a method named `getAnInteger()`, and modify it so that it loops until the user enters an integer. Additionally, your method also needs to print out the detailed exception. Here's an example interaction of a possible solution code:

```
Enter an integer: an integer
Here's what just happened: invalid literal for int() with base 10: 'an integer'
Enter an integer: 3.2
Here's what just happened: invalid literal for int() with base 10: '3.2'
Enter an integer: 32
Thanks! You entered the integer value 32
```

Your Task A.1 solution should match the above interaction as much as possible. (as usual, "hardcoded" interactions will be *rejected*).
*Hint*: use the `except` ... `as` statement.

2. [20 points]
Write another method named `readThisFile()`
Your method should prompt the user for a file name.
If the entered file name doesn't exist, you should *catch* the exception and ask for another file name.
Once the entered file name is valid, you should open the named file.
You may assume that the named file contains plain-text, saved as ASCII encoding (no UTF-8 or other encodings need to be handled), that each line of the file contains just one number in it, and no other content.
You should then print out the *sum* and the *average* of all number founds in the file, one number per line.

**Example Run**
In this example, there is a plain-text ASCII file named `my-numbers.txt` in the same directory as the Python program file, and the content of the text file is:

```
1
2
3
4
```

Here's an example interaction of a possible solution code:

```
Type a file name: my-numbers
A file with that name can't be found.
Type a file name: my-numbers.text
A file with that name can't be found.
Type a file name: my-numbers.txt
The numbers in your file "my-numbers.txt "add up to: 10.0
The average value of all numbers in your file is: 2.5
>>>
```

Your Task A.2 solution should match the above interaction as much as possible.

**Requirements**
Your solution needs to use at least two **helper methods**, otherwise it will be *rejected*. The two methods may be named as you prefer, but they need to satisfy the following:

- the first method needs to accept a string (the file name) as parameter, and return the list of numbers found in the named file.
- the second method needs to accept a list of numbers, and return two values: the sum and the average of all the numbers found in the list.

It is up to you to decide whether exception handling should be handled within these helper methods, or in the `readThisFile()` methods.
Again, "hardcoded" interactions will be *rejected*.

**Bonus**
Additional 5 points bonus will be awarded to any solution also handles exceptions caused by non-numeric content in the provided file. E.g. it should correctly handle file content such as:

```
1
2
three
3
4
```

*Note*: for full bonus credit, your solution should print out correct sum and average values, even when it finds non-numeric input such as the above example.

B. [50 points]
Modify the *High Scores 2.0* program (i.e. `high_scores2.py`) from Textbook Chapter 5 so that:
1. the program loads and saves the high scores using a text file called `scores.txt`.
2. When your program begins, it should attempt to load the high scores from the file.
3. If `scores.txt` doesn't exist, your program should use exception handling to avoid crashing, and it should display a warning that the file could not be opened.
4. When the user quits the program, your program should attempt to write the high scores to `scores.txt`, or create that file if it doesn't yet exist.
5. If your program is unable to write to the file (or create it), use exception handling to exit your program gracefully without crashing.

**Note about Task B:**
this task's purpose is to provide "persistent storage" for the High Scores 2.0 program*), i.e. making sure that if we quit the High Scores 2.0 program and restart it later, the program will remember any high scores we had entered so far.
To achieve this goal, your modified High Scores 2.0 program needs to save the high scores that have been entered by the user, into a file named `scores.txt`.
Then the next time the program is restarted, it should first attempt to read a file named `scores.txt` (if that file exists) and load any high scores it finds in the `scores.txt` file into the high scores list it maintains in a variable.
*the High Scores 2.0 program is one of the Python source code examples provided with the course textbook.
Please note the last item in the above list for Task B: your program needs to use exception handling for cases when opening or writing to the `scores.txt` file isn't possible. Such situations could happen for a number of reasons, e.g.: perhaps a file with that name already exists, but it is locked (in that case, your program won't be able to open the file for writing to it), or perhapes the file exists but it is already open for writing by another program (in that case, your program also won't be able to open the file for writing to it), or perhaps the hard disk is full (in that case, your program may be able to open the file, but it won't be able to write anything into the file), etc.

Save your solution to this task in a file named `pa13-highscores3.py`

C. [20 points total]
***Written Questions***
In answering these questions, it'll be helpful to review Reading Assignment 13.

1. Name at least four distinct exception types in Python, and explain when these exceptions are raised.
   [4 points]

2. How can you handle exceptions of different types that might arise from a single block of code?
   (i.e. explain at least two distinct ways your Python code can handle different types of exceptions arising from a single block of code)
   [4 points]

3. What are possible drawbacks in using "catch-all" exception handling in your Python code?
   [4 points]

4. How might you use an exception's argument?
   [4 points]

5. When should you consider trapping for exceptions?
   (provide at least two examples)
   [4 points]

Write your answers to the above questions in a plain-text file named `pa13-answers.txt`.
Make sure to number your answers appropriately, so that we know which answer belongs to which question.

## Programming Assignment 13 Submission Instructions:

When you submit your assignment on IU Canvas, it should consist of all the files detailed above.

### Rejection Warning

Please double-check the specific requirements for Task A and Task B above, so that your submission doesn't get rejected. Also, please remember course policies about turning in assignment files, in particular about Rejected Assignments: if you submit any part of this assignment in the wrong format, or if your Python script crashes, or if you are using version 2 of Python, then your assignment will be rejected and you will be given 0 points for the assignment until you fix the problems and resubmit it. If you are in doubt about whether you've got the correct version of Python or whether you're using the correct file formats, ask one of the instructors.

1. When you submit your Programming Assignment 13, also include a text file named **`pa13-log.txt`**, with your annotations as explained above.

2. When you submit your assignment on IU Canvas, it should consist of all the files detailed above..
   1. At the top of each one of your files, include "*A201 / Fall 2017*" (or "*A597 / Fall 2017*"), "*Programming Assignment 13*", your *full name* and your *IU username* (i.e. your IU account).
      Note: in your Python files (file extension .py), kindly include the same information at the top of the file, within Python `# comments`.
   2. Make sure that all your Python and plain-text documents are clearly readable by anyone; Python `.py` files, as all submitted plain-text documents, need to use either:
      - ASCII encoding (7-bit, allows no accents/smart quotes/etc.), ← preferred for Python source code submissions that only contain English-alphabet characters & symbols.
      - UTF-8 encoding (capable of encoding all sorts of characters) ← only for any submissions that *require* non-English-alphabet characters & symbols.

      (...otherwise your output may end up garbled...)
   3. Do not use any other document file formats: submitted files *other* than plain text (file formats such as .pdf, .doc, .docx, .html, .xml, etc...) will be *rejected*. In other words, all submitted `.txt` and `.py` files need to be plain text.
3. Turn in your Programming Assignment 13 files by 4:00PM on Monday, November 27, 2017, on IU Canvas.
4. When you turn in your programming assignment on IU Canvas, it is your responsibility to verify that your file(s) have been uploaded, and their content on the IU Canvas server.
5. Please do not omit your IU username from the content.

### Rejection Warning

Please double-check the specific requirements for the tasks above, so that your submission doesn't get rejected. Also, please remember course policies about turning in assignment files, in particular about Rejected Assignments: if your Python script crashes, if you submit any part of this assignment in the incorrect format, etc. then your assignment will be rejected and you will be given 0 points for the assignment until you fix the problems and resubmit it. If you are in doubt about whether you're using the correct file formats, ask one of the instructors.

## Getting Help

If you need help with Programming Assignment 13, or any other A201/A597-related topic:

- attend **office hours**, as listed on the Syllabus page
- ask the AIs, UIs, and Instructor:
  the recommended way to contact A201/A597 instructors is with a **Canvas message**.
- consult the textbook
- consult lecture notes
- consult the references provided in reading assignments

Last updated: 2017-11-16
mitja@indiana.edu