University of Saskatchewan
Department of Computer Science
176 Thorvaldson Building
110 Science Place, Saskatoon, SK, S7N 5C9, Canada
Telephine: (306) 966-4886, Facimile: (306) 966-4884

Assignment 3 – Solutions and Grading

Design, Testing and Debugging

Date Due: Wednesday, June 9, 11:59pm **Total Marks: 70**

Question 1 (30 points):

Five marks were added to this question for your testing. See "What to hand in" and "Evaluation".

Purpose: To practice planning, designing, implementing, and testing a Python program.

Degree of Difficulty: Moderate. Don't leave this to the last minute. The basic functionality of the various parts of this question are easy. There are aspects of some parts that you won't appreciate until you start testing.

References: You may wish to review the following chapters:

- Development Processes: PCS Chapter 5
- Procedural Abstraction: PCS Chapter 6.
- Testing: PCS Chapter 7.

Restrictions: This question is homework assigned to students and will be graded. This question shall not be distributed to any person except by the instructors of CMPT 145. Solutions will be made available to students registered in CMPT 145 after the due date. There is no educational or pedagogical reason for tutors or experts outside the CMPT 145 instructional team to provide solutions to this question to a student registered in the course. Students who solicit such solutions are committing an act of Academic Misconduct, according to the University of Saskatchewan Policy on Academic Misconduct.

Note: A3Q2 asks about your work on A3Q1

Keep track of how long you spend working on this question, including time spent reading the question, reviewing notes and the course readings, implementing the solution, testing, and preparing to submit. Also keep track of how many times you experienced unscheduled interruptions, delays, or distractions during your work; include things like social media, phone calls, texting, but not such things as breaking for a meal, exercise, commuting to or from the university. In A3Q2 you will be asked to summarize the time you spend working on A3Q1, and your answer will be more helpful to you if you have more precise records.

In this question you will apply an informal design process to the task of implementing a program at the level of Assignment 1. The purpose is to follow the process thoughtfully and reflectively. Please read the question description carefully before you start!

What to Hand In

Usually this appears last, but in this question, it's first so that you know what you have to hand in. As you can see, the code you develop is a small part of this work.

- Your plan to complete this question: `a3q1_plan.txt`
- Your design document: `a3q1_design.txt`
- Your answers to the reflection questions: `a3q1_reflections.txt`
- Your Python program `a3q1.py`
- **NEW!** Your test script `a3q1_testing.py`

You are allowed to use PDF, RTF, DOC files instead of TXT. Be sure to include your name, NSID, student number, course number and lecture section at the top of all documents.

Problem Specification

Restrictions

This question is homework assigned to students and will be graded. This question shall not be distributed to any person except by the instructors of CMPT 145. Solutions will be made available to students registered in CMPT 145 after the due date. There is no educational or pedagogical reason for tutors or experts outside the CMPT 145 instructional team to provide solutions to this question to a student registered in the course. Students who solicit such solutions are committing an act of Academic Misconduct, according to the University of Saskatchewan Policy on Academic Misconduct.

The program you will write will be concerned with word-chains, defined as follows. A word-chain is a sequence of words where each sequential pair of words differs by exactly one letter. For example, the following is a valid word-chain, consisting of exactly 3 words:

```
shop
stop
step
```

The words "shop" and "stop" differ by exactly one letter; the words "stop" and "step" differ by exactly one letter. Word-chains can be arbitrarily long.

However, the following is not a valid word-chain:

```
cold
coat
boat
```

It is not valid because the words "cold" and "coat" differ by more than one letter.

Your program will determine whether a word-chain is valid by the following criteria, in the following order:

1. **The word chain has at least 2 words in it.** If there are fewer than 2 words in the word-chain, your program should display the following text on the console:

   ```
   Invalid: too short
   ```

2. **Each word is a valid word.** If an invalid word (e.g., "sidx") appears in the word-chain, your program should display the following text on the console:

   ```
   Invalid word: sidx
   ```

   You only need to identify the first invalid word in the word-chain.

   To determine whether a word is valid, you will be given a file with 500 four-letter words, in alphabetical order, called `four.txt`. These words are valid; if any word in a word-chain does not appear in this file, it is not valid.

3. **No word in the word-chain appears more than once.** If a word appears more than one in the word-chain, your program should display the following text on the console:

   ```
   Invalid: repeated word: case
   ```

   You only need to identify the first repeated word in the word-chain.

4. **Each sequential pair of words differs by exactly one letter.** If two words in the word-chain differ by more than a single letter, your program should display the following text on the console:

```
Invalid step from bred -> arid
```

You only need to identify the first invalid step in the word-chain.

5. **If the word-chain is valid by the above criteria,** then your program should display the following text on the console:

```
Valid!
```

Word-chain examples are provided in a ZIP file. Each example is a file containing 4-letter words, one word per line. Your program should ask for the name of a file, read the file, and check for validity. For example, it might look like this:

```
Name of chain file to check? A3Q1_09.txt
Invalid step from fell -> call
```

Task

Restrictions

This question is homework assigned to students and will be graded. This question shall not be distributed to any person except by the instructors of CMPT 145. Solutions will be made available to students registered in CMPT 145 after the due date. There is no educational or pedagogical reason for tutors or experts outside the CMPT 145 instructional team to provide solutions to this question to a student registered in the course. Students who solicit such solutions are committing an act of Academic Misconduct, according to the University of Saskatchewan Policy on Academic Misconduct.

The purpose of this question is to get you to deliberately spend time planning, designing, implementing, and testing an application, using the concepts from Chapters 4-7. You will be expected to create a design document, along the lines of A2Q3, then implement the design, and test the implementation. To accomplish this, you should create a plan, in which you schedule the different parts of your work (see page 8).

For this question you will do the following:

- You will start by creating a plan for your work. Your plan will schedule time for the following development phases:
  - **Requirements.** The specification is given above, so this part of the plan includes reading the Specification section carefully. You do not need to hand in anything for this part.
  - **Design.** Your plan will indicate when you will do the design phase (day, time, location), and how long you think it will take (duration).
  - **Implementation.** Indicate when you will work on the implementation phase (day, time, location), using your design, and how long you think it will take (duration).
  - **Testing and debugging.** Indicate when you will work on testing and debugging (day, time, location), and how long you think it will take(duration).
  - **Final clean up for submission.** Indicate when you will work on clean-up (day, time, location), and how long you think it will take (duration).

  You must submit this plan for grading.
- You will design your program, producing a design document outlining your implementation. You will submit this design for grading.

- You will record the amount of time you actually spent on each phase of the plan. You will submit this information for grading.

Note: If you do not engage with the planning and design, you are avoiding the main learning objective. The application you are developing is not important enough to spend time on, except to gain some experience doing the planning and design.

Details about the Design Your design document should describe your implementation in terms of the functions you'll need to complete the program. For each function, you must specify:

- Pseudo-code or informal algorithm descriptions. The form of the description is up to you.
- An informal description of the inputs and outputs for each function.
- An informal description of any test cases you will need to check your functions.

It's important for you to keep in mind that the purpose of the design is to help you, and you should not be overly worried about the format of your design.

Details about the Implementation, Testing, and Debugging Try to apply an incremental approach. For each function in your design:

- Implement the function according to your design.
- Test your function.
- Debug as necessary.

Keep track of your time during these phases. You'll want some objective evidence for how long things actually take, compared to what you estimated in your plan. This objective evidence will help you prepare more realistic plans in the future! How you actually complete these phases is less important than being objective about your progress.

Details about your submitted program The program is the least important part of the assignment. Make it look presentable, by adding comments and doc-strings to your functions.

Details about your reflection Your answers to the reflection questions below are graded for relevance and thoughtfulness of your answer. There are no right answers, and the only way to lose these marks is to fail to reflect meaningfully.

Restrictions

This question is homework assigned to students and will be graded. This question shall not be distributed to any person except by the instructors of CMPT 145. Solutions will be made available to students registered in CMPT 145 after the due date. There is no educational or pedagogical reason for tutors or experts outside the CMPT 145 instructional team to provide solutions to this question to a student registered in the course. Students who solicit such solutions are committing an act of Academic Misconduct, according to the University of Saskatchewan Policy on Academic Misconduct.

Reflection questions

1. How long did it take for you to create your plan?
2. Did you revise your design? In other words, did any part of your implementation turn out to be significantly different from your design? If so, briefly explain the reason for the difference(s).
3. Did any of the phases take significantly longer than you had planned? If so, explain what happened, and suggest ways to avoid this in the future.

Evaluation

Restrictions

This question is homework assigned to students and will be graded. This question shall not be distributed to any person except by the instructors of CMPT 145. Solutions will be made available to students registered in CMPT 145 after the due date. There is no educational or pedagogical reason for tutors or experts outside the CMPT 145 instructional team to provide solutions to this question to a student registered in the course. Students who solicit such solutions are committing an act of Academic Misconduct, according to the University of Saskatchewan Policy on Academic Misconduct.

- 5 marks: Your plan allocated time to every phase of the development process, and the times were plausible.
- 5 marks: Your design gave complete and useful details in terms of function descriptions, pseudo-code algorithms, and test cases.
- 5 marks: Your answers to the reflection questions were thoughtful and relevant.
- 5 marks: Your program is correct, and not terribly inefficient.
- 5 marks: Your program is well-documented with helpful comments for the reader.
- **NEW!** 5 marks: Your test script has test cases for every function, and is plausibly thorough.

Development Plan

Phase	Day	Time	Duration	*Actual Time*
Requirements				
Design				
Implementation				
Testing/debugging				
Tidying up				

Solution:

Restrictions

This solution shall not be distributed to any person except by the instructors of CMPT 145.

First, let's start with a plan:

Development Plan

Phase	**Day**	**Time**	**Duration**	*Actual Time*
Requirements	Friday	morning	10m	10m
Design	Friday	morning	20m	18m
Implementation	Friday	morning	30m	27m
Testing/debugging	Friday	morning	30m	52m
Tidying up	Friday	morning	15m	3m
Totals			105m	110m

1. **How long did it take for you to create your plan?**

   This plan required me to think for about 2 minutes.

2. **Did you revise your design? In other words, did any part of your implementation turn out to be significantly different from your design? If so, briefly explain the reason for the difference(s).**

   My original design had a function to check if every pair of words in the word list differed by exactly one letter. But then I changed it to check only one pair. The main program now has to loop through the pairs of words, but testing the function is easier now. I also decided to add test cases for this pair-of-words function.

3. **Did any of the phases take significantly longer than you had planned? If so, explain what happened, and suggest ways to avoid this in the future.**

   I noticed that my estimate for the "Duration" is pretty close. But my testing of the main program using the example files took me a bit longer than I guessed, because of an error. It was a dumb error, caused by copy/paste. I didn't estimate enough time for testing, so I went over-time. Maybe the next time I will not be so optimistic.

The Solutions folder contains:

- The design of this application in the file `a3q1_DesignDoc.txt`. This design was created by top-down design, starting with a paraphrase of the assignment description (the 5 criteria). Then a function was designed for each one of these steps. Test cases were developed based on the design.
- The implementation of this application in the file `a3q1.py`. The functions are thoroughly documented, and correspond with the design, with one minor exception. See below!
- A testing script (test driver) in the file `a3q1_testing.py`.

Notes for markers: I think markers should only run code if they have doubts about the code. CMPT 145 hasn't covered enough Python to make running the code convenient for markers and students. Mark visually, and generously.

- 5 marks: Your plan allocated time to every phase of the development process, and the times were plausible.
  - Give 5 marks if a plan was submitted with times allocated to each phase. The times have to be plausible (not too long and not too short). The important parts of the plan are "Duration" (i.e., how much time planned on a phase), and "Actual time". The Day/Time columns are not as important.
  - Give 3 marks if some phases were not allocated any time, or if times were not plausible.
  - Give 0 marks if no plan was submitted. I expect a table like the one shown, but something simpler could be used.
- 5 marks: Your design gave complete and useful details in terms of function descriptions, pseudo-code algorithms, and test cases.
  - Give 5 marks if a design was submitted with specification of a number of functions. The design has to mention input/output (or precondition/return are acceptable). An algorithm is appropriate, but not necessary for full marks. Test cases are appropriate if included, but not necessary.
  - Give 3 marks if the design dos not specify functions that can be tested, or if the functions are not described with a purpose, and inputs/outputs.
  - Give 0 marks if no design was submitted.
- 5 marks: Your answers to the reflection questions were thoughtful and relevant.
  - Give 5 marks if the responses were plausible.
    1. The time to plan should not be long. Anything longer than 10 minutes is not plausible.
    2. Design revisions happen, but are not required. The question here is for students to learn from. Any relevant discussion here is okay.
    3. Time required for the phases. The question here is for students to learn from. Any relevant discussion here is okay.
  - Give 3 marks if the discussion was not relevant, or too superficial.
  - Give 0 marks if no answers were submitted.
- 5 marks: Your program is correct, and not terribly inefficient.
  - Give 5 marks if the code looks okay. Executing the code is not necessary. The implementation is not the point of this question.
  - Give 3 marks if you can detect errors in the code while reading it.
  - Give 0 marks if there was nothing submitted, or if the implementation was not close to being completed.
- 5 marks: Your program is well-documented with helpful comments for the reader.
  - Give 5 marks if each function in the program has a doc-string as required by Chapter 6.
  - Give 3 marks if some functions were not documented with the right kind of doc-string, of if more than a few doc-strings were incorrect, or incomplete.

  - Give 0 marks if there was little or no documenting of functions, or if the documentation was not in the style of Chapter 6.

- 5 marks: Your test script has test cases for every function, and is plausibly thorough.
  - Give 5 marks if every function was tested, except possibly for functions that only do console (or file) input/output tasks. Functions that return a value should be tested.
  - Give 3 marks if most functions were tested, or if all function were tested, but not very thoroughly. Also, if there is too much testing, or test cases that are redundant or random.
  - Give 0 marks if too little testing was done, including no test script at all.

Question 2 (12 points):

Purpose: To reflect on the work of programming for Question 1. To practice objectively assessing the quality of the software you write. To practice visualizing improvements, without implementing improvements.

Degree of Difficulty: Easy.

Textbook: Chapter 3

Restrictions: This question is homework assigned to students and will be graded. This question shall not be distributed to any person except by the instructors of CMPT 145. Solutions will be made available to students registered in CMPT 145 after the due date. There is no educational or pedagogical reason for tutors or experts outside the CMPT 145 instructional team to provide solutions to this question to a student registered in the course. Students who solicit such solutions are committing an act of Academic Misconduct, according to the University of Saskatchewan Policy on Academic Misconduct.

Answer the following questions about your experience implementing the program in Question 1 . You may use point form, and informal language. Just comment on your perceptions. Be brief. These are not deep questions; a couple of sentences or so ought to do it.

1. (2 marks) Comment on your program's *correctness* (see Chapter 3 of the textbook for the definition). How confident are you that your program (or the functions that you completed) is correct? What new information (in addition to your current level of testing) would raise your confidence? How likely is it that your program might be incorrect in a way you do not currently recognize?
2. (2 marks) Comment on your program's *efficiency* (see Chapter 3 of the textbook for the definition). How confident are you that your program is reasonably efficient? What facts or concepts did you use to estimate or quantify your program's efficiency?
3. (2 marks) Comment on your program's *adaptability* (see Chapter 3 of the textbook for the definition). For example, what if Assignment 2 asked you to write a program to check whether a 5×5 square was magic (bigger square with a larger sum, using the numbers 1 through 25)? How hard would it be to take your work in A1Q1, and revise it to handle squares of any size?
4. (2 marks) Comment on your program's *robustness* (see Chapter 3 of the textbook for the definition). Can you identify places where your program might behave badly, even though you've done your best to make it correct? You do not have to fix anything you mention here; it's just good to be aware.
5. (2 marks) How much time did you spend working on this program? Did you use the given implementation plan, or did you adjust the plan to suit your situation? Did it take longer or shorter than you planned? If anything surprised you about this task, explain why it surprised you.
6. (2 marks) Consider how often you were interrupted, distracted, delayed during your work for Question 1. Do you think these factors affected substantially increased the time you needed? If so, what kinds of steps can you take to prevent these factors? Were there any interruptions that you could not have prevented? How did you deal with those?

What to Hand In

Your answers to the above questions in a text file called `a3q2.txt` (PDF, rtf, docx or doc are acceptable). Be sure to include your name, NSID, student number, course number and laboratory section at the top of all documents.

Evaluation

The purpose of these questions is to reflect on your experience. You are not expected to give the "right answer", or to have worked with perfection. Your answers are for you. We will give you credit for attempting to use this opportunity to reflect in a meaningful way, no matter what your answers are.

Each answer is worth 2 marks. Full marks will be given for any answer that demonstrates thoughtful reflection. Grammar and spelling won't be graded, but practice your professional-level writing skills anyway.

Solution:

Restrictions

This solution shall not be distributed to any person except by the instructors of CMPT 145.

There are no wrong answers, but marks will be deducted if your answers are not relevant to the issues at hand. Here are some comments, and the level of discussion we would expect.

1. The test script suggests that the code is correct, with high confidence. I don't think adding more tests would increase my confidence. I also happened to implement the task twice, with slightly different designs. I got the same answers both times for all of the Example files.

2. My implementation is pretty efficient. I took care NOT to put all the words from the file `four.txt` into a list. Instead I used a dictionary. We haven't studied algorithm analysis yet, and so the efficiency difference between using a list and a dictionary might be hard to appreciate at this point in the course. However, we know that linear search in a list is rather expensive, especially for long lists. A Python dictionary, it turns out, can be much more efficient. Another option was to use binary search for a word through the acceptable words list. That would be as good as using a dictionary.

3. All of the functions in my design are already written to allow reuse for any size word, and any length of word chain. There would be almost no extra programming for longer words or word chains. If we wanted to add new criteria (or eliminate some), we could easily make the modifications. So I would say it's highly adaptable.

4. My implementation makes no effort to be robust. My program will crash if:

   - The file `four.txt` were missing.
   - The file name entered by the user does not exist.

5. The implementation took about 2 hours. I estimate it would have taken less time to write without a design document, but I would have taken more short-cuts, and I would not have had such neat functions to test. Debugging might have required more of my time, even if I spent less time writing code.

6. During the time I wrote up these solutions, I was able to work mostly without interruptions. The phone rang once, and I had to take care of short tasks in the laundry. I don't use social media, except for Discord and Moodle related to my teaching, and these were pretty quiet during this work.

Notes for markers:

- Give zero marks here if no solution was submitted for Q1.
- This question is for students to practice thinking about these issues. There is no need for the answers to be positive. If the student knows their code is not very good, and says why, that's worth full marks here.
- Give zero marks for any part if no response was given, or if the response was completely irrelevant.
- Give one mark if the response seemed too superficial. A superficial response would be something like: "I'm sure my program is correct" without referring to test cases.
- CMPT145 students have not yet been taught anything formal about complexity analysis, so discussion on efficiency will be general and somewhat vague, and even mistaken! The point is to consider the questions thoughtfully.

Question 3 (28 points):

Purpose: To practice writing doc-strings. To practice thinking about test-case design.

Degree of Difficulty: Easy.

Restrictions: This question is homework assigned to students and will be graded. This question shall not be distributed to any person except by the instructors of CMPT 145. Solutions will be made available to students registered in CMPT 145 after the due date. There is no educational or pedagogical reason for tutors or experts outside the CMPT 145 instructional team to provide solutions to this question to a student registered in the course. Students who solicit such solutions are committing an act of Academic Misconduct, according to the University of Saskatchewan Policy on Academic Misconduct.

The file `a3q3.py` contains three functions, most of which are correct (but some may not be). In this question, you will be guided in the preparation of a test script to test the functions.

- `newton()`. This function implements a well-known algorithm for computing the square root of a non-negative number.
  1. Study the function, and write an appropriate doc-string for it.
  2. Design test cases for this function. There are 2 equivalence classes that are of interest.
     - Numbers between 0 and 1. The square root is bigger than the input number.
     - Numbers greater than 1. The square root is smaller than the input number.
     - There are two boundary cases: 0, and 1.

     Write test cases for this function, considering the information above. Don't just test integers!
- `euclid()`. This function implements a well-known algorithm for computing the greatest common divisor of two integers. For example, $\mathrm{euclid}(27, 6) = 3$, since 3 is the largest integer that evenly divides 27 and 6.
  1. Study the function, and write an appropriate doc-string for it.
  2. Notice that the function prints to the console. It's a terrible design decision; it's nearly impossible to test console output from within Python. Before you test the function, modify the function to return the value instead.
  3. Design test cases for this function. There are several equivalence classes of interest:
     - $a > b$
     - $b > a$
     - $a = b$
     - What happens when $a = 0$ or $b = 0$? Is this okay? You may need to do some research to figure this out.

     Write test cases for this function, considering the information above.

- `pascal()`. This function opens a named file and reads the contents. An example of the kind of file that it reads is given, named `triangle1.txt`. Notice that the first line in the file is the length of the example.
  1. Study the function, and write an appropriate doc-string for it.
  2. Create a test script for this function. Testing a function like this is difficult because you don't want to type in correct result in your test script. It is hard to do test functions that read from a file completely. So here is a strategy that will help:
     - The function reads "Pascal Triangles". Create a couple more example files for testing. Smaller examples are fine!
     - Test that the returned `size` is correct for the example files.
     - Test that the length of the returned `triangle` is correct.
     - Test that the elements of the triangle are also lists. To check if `alist` refers to a list, you can use the Python code `type(alist) is list`. This is a Boolean expression that returns `True` if alist was constructed by the list constructor. It will be `True` for lists, and `False` for everything else.

     Write test cases for this function, considering the information above.
- `turing()`. This function removes duplicates from a given list, keeping only the first occurrence of any duplicated value. For example, given the list `[1, 2, 3, 1]`, `turing()` will remove the second occurrence of 1.
  1. Study the function, and write an appropriate doc-string for it.
  2. Create a test script while considering the following equivalence classes:
     - No duplicates.
     - One duplicated item.
     - One duplicated item, with duplicates appearing sequentially.
     - Multiple duplicated items appearing anywhere in the list.
     - The only boundary case is the empty list.

     Write test cases for this function, considering the information above.

Restrictions

This question is homework assigned to students and will be graded. This question shall not be distributed to any person except by the instructors of CMPT 145. Solutions will be made available to students registered in CMPT 145 after the due date. There is no educational or pedagogical reason for tutors or experts outside the CMPT 145 instructional team to provide solutions to this question to a student registered in the course. Students who solicit such solutions are committing an act of Academic Misconduct, according to the University of Saskatchewan Policy on Academic Misconduct.

What to Hand In

- A Python script named `a3q3_newton.py` with your test script for `newton()`.
- A Python script named `a3q3_euclid.py` with your test script for `euclid()`.
- A Python script named `a3q3_pascal.py` with your test script for `pascal()`.
- A Python script named `a3q3_turing.py` with your test script for `turing()`.
- A Python script named `a3q3.py` with the functions. You've added doc-strings, and modified `euclid()` as indicated.

Be sure to include your name, NSID, student number, section number, and laboratory section at the top of all documents.

Evaluation

Restrictions

This question is homework assigned to students and will be graded. This question shall not be distributed to any person except by the instructors of CMPT 145. Solutions will be made available to students registered in CMPT 145 after the due date. There is no educational or pedagogical reason for tutors or experts outside the CMPT 145 instructional team to provide solutions to this question to a student registered in the course. Students who solicit such solutions are committing an act of Academic Misconduct, according to the University of Saskatchewan Policy on Academic Misconduct.

- Function interface documentation. You added the doc-strings giving the correct information.
  - 2 marks for each function.
- Your test scripts perform the indicated testing.
  - 5 marks for each function. Marks will be deducted if
    - the testing does not include sufficient test cases
    - some of the equivalence classes are not tested.
    - the test script is overly convoluted or otherwise demonstrates poor style or design

Solution: The full solution can be found in the files:

- a3q3.py
- testing_newton.py
- testing_euclid.py
- testing_pascal.py
- testing_turing.py

The test scripts should contain a solid collection of test cases. The assignment description explained what test cases to try, and how many. There were a few issues that came up.

newton()

```
1  def newton(x):
2      """
3      Purpose
4          Computes the square root of a non-negative number
5      Preconditions:
6          :param x: a non-negative number
7      Post-conditions:
8          None
9      Return
10         :return: a number
11     """
```

The issue here is floating point equality. A number of methods will have been tried to check for equality, since the function returns an approximation only. Any attempt to deal with the difference between the output of the function and a known value is acceptable. The preferred way is to compare the numbers to see if they are close enough. This idea is presented in the test script, and also part of Assignment 4.

euclid()

```
1  def euclid(a, b):
2      """
3      Purpose
4          Calculates the greatest common divisor of two numbers
5      Preconditions:
6          :param a: integer must be greater than 0
7          :param b: integer must be greater than 0
8      Post-conditions:
9          None
10     Return
11         :return: the biggest positive integer that evenly divides both a and b
12     """
```

There were two issues here: finding the actual GCD for a pair of numbers, and the problem of zero as input arguments. Students may have included test cases that used zeros as input. This causes an infinite loop. The right thing to do is to check the input values for zero, and avoid starting the loop in that case. However, students were not required to fix the functions. Leaving the zero input test cases in the script is not ideal, but acceptable. Commenting them out is acceptable too. And if they were not used at all because of the infinite loop problem that's okay, too. I am hoping students experienced the problem, but their grade should not depend on how they addressed it.

pascal()

```
1  def pascal(filename):
2      """
3      Purpose
4           Reads a file and puts the contents into a list.
5      Preconditions:
6          :param filename: a string
7      Post-conditions:
8          None
9      Return
10         :return: a tuple (size, data)
11     """
```

Testing a function like this is tricky, especially if you have a mental block about what testing actually is. In the simplest unit cases, we can test a function's output against a known value. In this case, the known value is quite complicated, and the value of typing that known value into a test script is very low. Don't learn much, and spend a lot of time. The assignment description tried to suggest that a few indirect tests of the properties of the return values could be an adequate replacement. Students may or may not have submitted their additional example files. They were encouraged to create a couple.

turing()

```
1      """
2      Purpose
3           Removes duplicates in the given list
4      Preconditions:
5          :param alist: a list
6      Post-conditions:
7          duplicate values are removed
8      Return
9          :return: None
10     """
11     alist.reverse()
```

This function is broken. It's actually a reconstruction of an example I helped a student debug in previous offerings of the course. The test cases however, are straight forward. They're annoying to write, but not hard to come up with given the guidance in the assignment. These test cases will reveal the presence of errors in the function, but again, there was not requirement to fix the code.

Notes for markers:

- Doc-strings must have all 4 components: Purpose, Pre-conditions, Post-conditions, Return. These are explained in Chapter 6. For this assignment, deduct 1 mark per function if something is missing or not done well. Hopefully, the information is more or less correct, but this time, we'll ignore any misunderstandings about what information to provide.
- Testing: The question asked for 2 examples from each of the equivalence classes, and a few boundary cases. Give full marks for a solid effort for each script. If a submission has less than half of the recommended testing, deduct 2 marks for that function. Reserve the minimal grade for seriously deficient efforts.