CS1P Lab exam

Lab Exam 1 2016/2017

- **28th-30th November 2016**
- **Time allowed:** 1hr50 minutes + preparation time

This exam is worth 10% of the marks for the CS1P module.

Students must attempt at least 75% of all assessments in CS1P to get credit.

Read the following very carefully

Instructions

Before the exam

You should prepare, test and thoroughly understand a complete solution to the exam question in advance of the exam. Put in as many hours as you need to get a properly working solution that you fully understand.

In the exam

When you come into the exam, you can bring nothing with you. You will get another copy of the question along with some rough working paper. You need to be prepared to reconstruct your solution to the problem during the exam.

You may of course talk to others while preparing for the exam.

Remember, however, that you will be on your own in the exam, and so you must understand your solution thoroughly before coming to the exam.

Network

During the exam, the machines will be disconnected from the network and from your filestores. You will have no access to the web or any internet resources.

Access

There will no access to the laboratory, *other than to take the lab exam*, at any time during the period 28-30th November inclusive. The lab may also be closed for short periods between 11.00 and 14.00 during the Thursday/Friday before to allow for preparation by the systems team.

Starting the exam

Complete the attendance slip and place it in front of you together with your matriculation card.

You must bring your matriculation card to the exam.

Log in to the system using your *special exam username and password* as provided on the document in your examination pack. *Don't use your standard login; it won't work.*

In the exam: AMS

[Note: this only applies when you are in the exam itself; while preparing you will have access to the exam via Moodle/your filestore]

- Double-click the AMS icon on the desktop.
- You should see LabExam1 in the drop-down box at the top-right.
- Press the **Setup** button.
- This will generate a copy of the template program file in your exam account Workspace. It will be the same as the template you downloaded from Moodle for the preparation.
- Launch IPython from the desktop, as usual.
- Open `lab_exam.ipynb`

During the exam

As in a normal examination, **communication between candidates during the laboratory exam is strictly forbidden.** This is considered cheating, and if you are caught cheating, the university response will have dire consequences for you. Don't risk it.

You may leave early, but only after summoning an invigilator, who will ensure that submission has taken place. If you experience a hardware or software problem during the examination should summon an invigilator *at once.*

At the end of the exam

Ensure you have submitted your solution, using the AMS system, before the end of the examination.

In the exam: AMS submission

- Reopen AMS if you have closed it
- Press the **Submit** button

- You are done.

Leaving the exam

Leave the laboratory quietly, as the exam may still be in progress for other groups and/or special needs candidates.

Problem

Context

Compression utilities such as ZIP make use of the repetitive nature of text. Compression utilities find short ways of writing text by taking advantage of **repetitive structure**. For example, it is fairly obvious that

```
00000000000000000000000000000000
```

is somehow simpler than

```
e8uifgtw90ef8*£(*()£*()£jfkplmwf£*(
```

and could perhaps be written with a some shorter code.

You will build a real, effective compression algorithm, which is known as *LZW* compression. It is a simple and clever algorithm, which works by *remembering sequences seen before* and giving them shorthand codes.

To do this compression, you will have to transform text into a sequence of integer codes. LZW operates on strings, and it outputs a sequence of integer codes. For example, "ABC" might become `65, 66, 67` when encoded; "AAAAA" might become `65, 128, 128` (note that only three numbers are used to encode five characters).

The LZW algorithm

The algorithm to compress a string is as follows:

- Assign an initial integer code to each possible character. This is the initial **codebook**
- Keep a **buffer** of the last remembered substring, which is initially empty
- Iterate through each **character** of the string to compress.
    - Keep appending **characters** to the **buffer** until there is no **codebook** entry for the **buffer**. When this happens:
        - Add a **new code** for the **buffer** to the codebook.

    - Append the integer code for the **buffer** *without the last character* to the compressed output. By definition, this code must be in the codebook.
    - Clear the **buffer**.

Output

The result of this compression will be two data structures:

- a sequence of integers representing the compressed version of the string. (**the compressed output**)
- a mapping from strings to integer codes (**the codebook**).

Make sure your compressor correctly compresses *all* of the string

Decompression

The decompression is straightforward given the codebook. It is actually possible to reconstruct the string *without* being given the codebook, but for this exercise you may assume your decompresser will receive the codebook that the compressor created.

Problem specification

You are to write two functions:

```
compress
    Takes an uncompressed string, and compresses according
    to the compression scheme given.
    Returns the compressed result, and the codebook.

decompress
    Takes a compressed version and the corresponding codebook.
    Returns the uncompressed string.
```

You should verify in your testing that

```
compressed, codebook = compress(s)
decompress(compressed, codebook) == s
```

is `True` regardless of the value of `s`; i.e. that compression then decompression does *not* alter the string in any way. Your compressor must work for a string that may contain any ASCII characters.

Hint:

- Remember that every ASCII character already has a unique integer code.
- Consider what adding a new code means (it must be a number not already used in the codebook).

- Your integer codes should be sequential (i.e. without gaps).
- Write `compress` first; `decompress` is hard to test properly without writing a working compress function.

You will have to choose sensible data structures for the compressed data and for the codebook, and implement the LZW compression and decompression algorithms correctly.

You will get 60% of the marks for `compress` and 40% for `decompress`.

Sample input and output

There are a series of cells in the problem notebook provided that will test your solution. You should verify that these tests work correctly. There will be additional testing applied **after you have submitted your solution.** You will not be able to see these additional tests.

Availability of functions

Do not use any external modules. There should be no import statement in your code.
You should use constructs only from those covered in the course so far; i.e. those that appeared in the lecture notes. You can be penalised for not following this instruction.

Notebook structure

Only the file `lab_exam.ipynb` will be collected by the AMS system. Make absolutely sure your solution is in it.

It is vital that you do not alter or remove any part of the notebook, other than editing your solutions for compress and decompress.

Making copies

During the lab exam, make copies of your notebook regularly (for example, every 30 minutes), using the menu option `File/Make a Copy`. It will be *your* responsibility if something goes wrong and you don't have a backup copy handy!

If you do make a copy and need to recover from it, you must copy your work back into the original notebook (`lab_exam.ipynb`) Don't just start working in the copy or your work will not be seen.

Marking guideline

There are 20 marks available for this lab exercise.

The `compress` function counts for 12 of these marks. The `decompress` function counts for 8 marks.

The following aspects of the program will be taken into account in marking

- use of appropriate and correct algorithms
- use of appropriate data types
- use of appropriate Python control structures
- correct Python syntax
- good programming style, including layout, use of explanatory comments, choice of identifiers, and appropriate use of functions
- correctness of the implementation (as indicated by outputs corresponding to secret test inputs)

If your program does not work correctly, it can only acheive a maximum of 15 marks.