Student Number

The University of Melbourne
School of Computing and Information Systems

Final Examination, Semester 2, 2017
COMP10001 Foundations of Computing

Reading Time: 15 minutes. **Writing Time:** 2 hours.

This paper has 17 pages including this cover page.

Instructions to Invigilators:

Students must write all of their answers on this examination paper. Students may not remove any part of the examination paper from the examination room.

Instructions to Students:

There are 10 questions in the exam worth a total of 120 marks, making up 50% of the total assessment for the subject.

- All questions should be answered by writing a brief response or explanation in the lined spaces provided on the examination paper.
- It is not a requirement that all the lined spaces be completely filled; answers should be kept concise. Excessively long answers or irrelevant information may be penalised.
- Only material written in the lined spaces provided will be marked.
- The reverse side of any page may be used for notes or draft answers.
- Your writing should be clear; illegible answers will not be marked.
- Extra space is provided at the end of the paper for overflow answers.
  Please indicate in the question you are answering if you use the extra space.
- Your answers should be based on Python 3 (the version that Grok uses), and can use any of the standard Python libraries.

Authorised Materials: No materials are authorised.

Calculators: Calculators are not permitted.

Library: This paper may be held by the Baillieu Library.

Examiners' use only

1	2	3	4	5	6	7	8	9	10	Total

Part 1: Algorithmic Thinking

Question 1

[10 marks]

Evaluate the following expressions, and provide the output in each case.

(a) `'green' + 'eggs'[-1]`

(b) `sorted({'donkeys':3, 'chickens':8}.items())[-1][1]`

(c) `('pi' in 'sky') or ('3.14'.isdigit())`

(d) `[x**2 for x in range(3,1,-1)]`

(e) `'winter is coming'.split()[0][1:4]`

Question 2

[10 marks]

Rewrite the following function, replacing the `while` loop with a `for` loop, but preserving the remainder of the original code structure:

```
def get_odd_squares(nums):
    odd_squares = []
    i = 0
    while i < len(nums):
        num = nums[i]
        square = num * num
        if (square % 2):
            odd_squares.append(num)
        i += 1
    return odd_squares
```

Question 3

[12 marks]

An "abecedarian" word is one where all of its letters are in alphabetical order, such as DEEP, ACCENT or ALMOST.

The following code is intended to be a function `reverse_abcd(word)` that determines if an all uppercase `word` is a *reverse* abecedarian word; that is, if all of its letters appear in *reverse* alphabetical order. Examples of reverse abecedarian words are SPOONFEED, WRONGED and ROOKIE.

```
1  def reverse_abcd(word):
2      result = True
3      last = 'Z'
4      i = 0
5      while i < len(word):
6          c = word[i]
7          if c < last
8              result = False
9          i += 1
10     last = c
11     return result
```

However, there are several errors in the given function definition. Identify exactly **three (3)** errors and specify: (a) the line number where the error occurs; (b) the type of error, as *syntax, logic* or *runtime*; and (c) how you would fix each error, in the form of corrected code.

This page is intentionally left blank

Question 4

[10 marks]

The function `repeat_word_count(text, n)` takes a string `text` and a positive integer `n`, converts `text` into a list of words based on simple whitespace separation (with no removal of punctuation or changing of case), and returns a sorted list of words that occur `n` or more times in `text`. For example:

```
>>> repeat_word_count("buffalo buffalo buffalo buffalo", 2)
['buffalo']
>>> repeat_word_count("how much wood could a wood chuck chuck", 1)
['a', 'chuck', 'could', 'how', 'much', 'wood']
```

As presented, the lines of the function are out of order. Put the line numbers in the correct order and introduce appropriate indentation (indent the line numbers to show how the corresponding lines would be indented in your code).

```
1   words[word] += 1
2   if words[word] >= n:
3   words = {} ; word_list = []
4   word_list.append(word)
5   def repeat_word_count(text, n):
6   words[word] = 0
7   return(sorted(word_list))
8   for word in words:
9   for word in text.split():
10  if word not in words:
```

Part 2: Constructing Programs

Question 5

[10 marks]

Consider a representation of a cargo terminal in which a number of *containers* are stored. Each container in the cargo terminal is either stacked on top of another container, or is on the ground. We can represent the *state* of the cargo terminal as a list. Each element of that list is itself a list of two strings `[s1, s2]`, where `s1` is the name of a container, and `s2` is the name of another container or `'ground'`, such that container `s1` is stacked on `s2`.

For example, `[['c1', 'c3'], ['c2', 'ground'], ['c3', 'ground']]` corresponds to the state in which containers `c2` and `c3` are sitting on the ground, container `c1` is stacked on top of `c3`, and containers `c1` and `c2` are clear, i.e., they have nothing stacked on top of them.

The function `count_clear(state)` takes a state of the cargo terminal, and returns an integer count of the number of containers in the given state that are currently clear (i.e., have nothing stacked on top of them). Note that the ground is not a container, so it does not count. You can assume there are no duplicates in `state`.

For example,

```
>>> count_clear([['c1', 'c3'], ['c2', 'ground'], ['c3', 'ground']])
2
>>> count_clear([['c1', 'c3'], ['c3', 'c2'], ['c2', 'ground']])
1
```

Provide code to insert into each of the numbered boxes in the code below to complete the function as described. Note that your code will be evaluated at the indentation level indicated for each box.

```
def count_clear(state):
    countclear = [    1    ]
    for tStack [    2    ]
        if [    3    ]
            countclear = [    4    ]
    [    5    ]
```

(1) ______________________________

(2) ______________________________

(3) ______________________________

(4) ______________________________

(5) ______________________________

Question 6

[14 marks]

The aim of this question is to implement a function that determines the next member in a *look-and-say sequence*.

A look-and-say sequence is a sequence of integers that begins as follows:

1, 11, 21, 1211, 111221, 312211, 13112221, 1113213211, . . .

To generate the next number in the sequence given the previous number, read the digits of the previous number, counting the number of digits in groups of the same digit. For example:

- 1 is read as "one 1" or 11
- 11 is read as "two 1s" or 21
- 21 is read as "one 2, then one 1" or 1211
- 1211 is read as "one 1, then one 2, then two 1s" or 111221
- 111221 is read as "three 1s, then two 2s, then one 1" or 312211

Write a function `look_and_say(prev)` that takes an integer `prev` (containing a member of the look-and-say sequence) and returns, as an integer, the next member of the look-and-say sequence. You may assume that `prev` is a valid member of the look-and-say sequence; i.e., an integer ≥ 1.

This page is intentionally left blank

Question 7

[18 marks]

You are given a table of integers (the `matrix`) and a table of booleans (the `template`).

Each row of these tables is stored as a tuple, and an entire table is stored as a tuple of rows; that is, a tuple of tuples.

Each element in the `matrix` is an integer in the range $[0, 9]$ (i.e., between 0 to 9 inclusive). For example, the following is an example of a 3-by-3 `matrix`:

```
matrix = ((0,1,3), \
          (2,4,1), \
          (6,0,5))
```

Each element of the `template` is a boolean (i.e., `True` or `False`). The width and height of the `template` will always be less than or equal to the width and height, respectively, of the `matrix`.

For example, the following is an example of a 2-by-2 `template`:

```
template = ((True, False), \
            (True, True))
```

Your aim is to position the `template` within the `matrix` in such a way that the sum of the elements in the `matrix` matching the `True` elements of the `template` is maximised. A position for the template is specified by a tuple `(row, col)`, where `row` and `col` are the row and column indexes in `matrix` at which the top-left element of `template` is positioned.

For example, in the example above, there are four valid positions for the `template` within the `matrix`: (0,0), (0,1), (1,0) or (1,1). It could not be positioned at, for instance, (0,2) as the right hand side of the `template` would then extend past the right hand side of the `matrix`.

The sum of matched elements in each case would be:

- (0,0): 0 + 2 + 4 = 6
- (0,1): 1 + 4 + 1 = 6
- (1,0): 2 + 6 + 0 = 8
- (1,1): 4 + 0 + 5 = 9

Therefore, the position (1,1) maximises the sum of matched elements.

Write a function `max_match(matrix, template)` that takes as input a `matrix` and `template` of arbitrary size (subject to the constraints outlined above), and returns a tuple containing (a) the maximum sum; and (b) a nested tuple containing the `template` position that produces this sum. In the example above, the return value would be `(9, (1,1))`.

```
>>> max_match(((0,1,3),(2,4,1),(6,0,5)), ((True,False),(True,True)))
(9,(1,1))
```

If multiple `template` positions would result in equal sums, then you may return *any one* of these positions.

Part 3: Conceptual Questions

Question 8: Algorithmic Problem Solving

[12 marks]

(a) In the context of algorithmic development, briefly explain what is what is meant by *heuristic search*?

[6 marks]

(b) Is *linear search* an example of heuristic search? Briefly explain why or why not.

[6 marks]

This page is intentionally left blank

Question 9: Applications of Computing [12 marks]

(a) Briefly describe **two** applications where *data mining* is used in practice.

[6 marks]

(b) Briefly explain the difference between *supervised learning* and *unsupervised learning* in the context of data mining.

[6 marks]

Question 10: HTML and the Internet

[12 marks]

(a) List **three** types of common *HTML elements* that can appear in the *body* of an HMTL document.

[6 marks]

(b) Briefly explain the difference between *static* and *dynamic* HTML pages.

[6 marks]

This is blank space for further answers should you need it. Please ensure that you label the answers in this area carefully, and that you indicate on the corresponding question page that your answer can be found here.

This page is intentionally left blank

This page is intentionally left blank

— End of Exam —