

Student Number

The University of Melbourne
Department of Computing and Information Systems

Final Examination, Semester 1, 2016
COMP10001 Foundations of Computing

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

This paper has 21 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 11 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.
- 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.

<i>Examiners' use only</i>											
1	2	3	4	5	6	7	8	9	10	11	Total

Part 1: Algorithmic Thinking

Question 1

[10 marks]

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

(a) `"brouhahas"[-5:-1] + "!"`

(b) `4/1`

(c) `bool("c" or "i" > "e")`

(d) `len({"11": "racehorse", "22": "ditto"}.items())`

(e) `[n for n in range(4) if n**2 > n]`

Question 2

[9 marks]

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

```
def dedup(lst):
    dedup_list = lst[:1]
    for i in range(1, len(lst)):
        if lst[i] != lst[i - 1]:
            dedup_list.append(lst[i])
    return dedup_list
```

[illegible]

Question 3**[10 marks]**

The function `shorten` is intended to take a string and shorten it, by reducing the lengths of the individual words (excluding punctuation) down to a maximum length by removing any “extra” characters in the middle. For example:

```
>>> shorten("In a hole in the ground there lived a hobbit.")
'In a hole in the grnd thre lied a hoit.'
```

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  for word in text.split():
2  break
3  def shorten(text, MAXLEN=4):
4  if word[-1] in '.,!?:':
5  word = word[:MAXLEN//2] + word[-MAXLEN//2:]
6  word, punct = remove_punct(word)
7  punct = ''
8  punct = word[-1] + punct
9  while word:
10 def remove_punct(word):
11 short_text = ''
12 short_text += word + punct + ' '
13 return word, punct
14 else:
15 if len(word) > MAXLEN:
16 word = word[:-1]
17 return short_text[:-1]
```


[illegible]

Question 4

[9 marks]

The following function is meant to take an integer `num` and decompose it into k -digit sub-sequences (noting that the first integer could be made up of less than k digits), map each sub-sequence back into a character based on its code point value, and compose the characters into a string. The following is an example function call which illustrates its intended behaviour:

```
>>> print(num2txt(97097114103104))
aargh
```

Identify exactly three (3) errors in the code (using the provided line numbers), determine for each whether it is a “syntax”, “run-time” or “logic” error, and provide a replacement line which corrects the error.

```

1  def num2txt(num, k=3):
2      numstr = str(num)
3      txt = ""
4      mismatch = numstr % k
5      if mismatch:
6          numstr = "0" * (k - mismatch)
7      start == 0
8      for end in range(k, len(numstr)+1, k):
9          txt += chr(int(numstr[start:end]))
10         start = end
11     return txt

```

This image shows a blank sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins or other markings on the paper.

Question 5**[10 marks]**

Each of the following Python code snippets has a bug in it, which leads to an exception being raised when run. For each code snippet, provide the exception type, from among the following:

- `AttributeError`
- `IndexError`
- `KeyError`
- `SyntaxError`
- `TypeError`
- `ValueError`

(a) `len(2)`

(b) `(3, 2, 1).sort()`

(c) `sorted(2, 'two', 2.0)`

(d) `max([])`

(e) `{'key': 'val', 'key2'}`

Question 7**[10 marks]**

The code on the next page is designed to analyse the number of cards in play, the size of the deck, and the timing of reshuffles of the deck in the context of a game of Oh Hell!, based on the rules used in Project 3 of this subject. Recall the rules of the game, as relevant to this question:

1. there are 4 players, and the game is played with a standard deck of 52 cards
2. for phases $N = \{1, \dots, 10\}$, each player is dealt N cards from the “deck”, and for phases $N = \{11, \dots, 19\}$, each player is dealt $20 - N$ cards from the “deck”
3. in each phase, one additional card is drawn from the deck to determine “trumps”
4. on completion of each phase, all cards dealt to players plus the single “trumps” card are put in a “discard” pile
5. when the cards in the “deck” are exhausted as part of the deal for a given phase, the “discard” pile is reshuffled and becomes the new deck

When run, the code should produce the following output, and calculate for each phase: (1) the total number of cards that are required for the deal; (2) the size of the “deck” after dealing; and (3) whether a reshuffle of the “discard” pile takes place in that phase or not:

phase	cards to deal	size of deck after dealing	reshuffled?
1	5	47	
2	9	38	
3	13	25	
4	17	8	
5	21	31	yes
6	25	6	
7	29	23	yes
8	33	19	yes
9	37	15	yes
10	41	11	yes
11	37	15	yes
12	33	19	yes
13	29	23	yes
14	25	27	yes
15	21	6	
16	17	35	yes
17	13	22	
18	9	13	
19	5	8	

Provide a single (non-complex) line of code to insert into each of the numbered boxes in the code to complete the function as described. Note that your code will be evaluated at the indentation level indicated for each box.


```

def card_count(midphase=10, deck_size=52, players=4):
    phase = cards = 1
    
    deck = deck_size
    total_dealt = 0
    reshuffled = ''
    row_fmt = "| {:>5s} | {:>13s} | {:>26s} | {:>11s} |"
    print(row_fmt.format("phase", "cards to deal",
                          "size of deck after dealing", "reshuffled?"))

    
    dealt = cards * players + 1
    deck -= dealt
    if deck < 0:
        
        reshuffled = 'yes'
    else:
        reshuffled = ''
    print(row_fmt.format(str(phase), str(dealt), str(deck), str(reshuffled)))

    
    ascending = False
    phase += 1
    
    cards += 1
    else:
        cards -= 1

card_count()

```

- (1) _____
- (2) _____
- (3) _____
- (4) _____
- (5) _____

Question 8**[18 marks]**

A “keisuke” puzzle takes the form of a rectangular grid with certain cells blacked out, and two sets of numbers made up of two or more digits, one denoted “across” and the other “down” (both of which are guaranteed to not contain duplicates). The objective is to fill the empty cells in the grid by placing a single-digit number in each cell such that the two-digit or longer numbers across the rows (left to right, not crossing any of the blacked-out cells) and down the columns (top to bottom, again not crossing any of the blacked-out cells) match exactly those in “across” and “down”, respectively. For example, given the following keisuke rectangle:

and the following sets of numbers:

- across: 23, 131, 233, 3221, 212223
- down: 12, 21, 22, 31, 232, 3132, 33313

the unique solution is:

2	3		1	3	1
	3	2	2	1	
2	3	3		3	
2	1	2	2	2	3
	3		1		1

We will represent keisuke rectangles using a two-dimensional tuple representation, with each element being either an integer value or `None` in the case of a blacked-out cell. In the case of the example above:

```
keisuke = ((2, 3, None, 1, 3, 1),
           (None, 3, 2, 2, 1, None),
           (2, 3, 3, None, 3, None),
           (2, 1, 2, 2, 2, 3),
           (None, 3, None, 1, None, 1))
```

Note that the number of rows and columns will vary according to the size of the rectangle.

The across and down numbers are supplied as tuples of integers; in the case of the example above:

```
across = (23, 131, 233, 3221, 212223)
down = (12, 21, 22, 31, 232, 3132, 33313)
```


Write a function `solved(keisuke, across, down)` that takes a keisuke rectangle, and across and down numbers as arguments, and returns `True` if keisuke is a solved puzzle based on across and down, and `False` otherwise.

For example:

```
>>> solved(((1, 1), (None, 3)), (11,), (13,))
True
>>> solved(((1, 1), (2, 3)), (11, 23), (12, 13))
True
>>> solved(((1, 1), (2, 3)), (11, 32), (12, 13))
False
```

In the first example, the keisuke rectangle is as follows:

1	1
	3

and the numbers to be filled are:

- across: 11
- down: 13

As such, it is a solved keisuke puzzle.

In the last example, the keisuke rectangle is as follows:

1	1
2	3

and the numbers to be filled are:

- across: 11, 32
- down: 12, 13

The two across numbers in the actual puzzle are 11 and 23, and don't match those to be filled (11 and 32), meaning that it is not a correctly solved keisuke puzzle.

Note that you may assume that `keisuke` is well-formed (i.e. a non-empty rectangular tuple of tuples, with each value being a single-digit number or `None`), and similarly that `across` and `down` are well-formed tuples of non-negative integers without duplicates, each of which is made up of at least 2 digits. Note also that you may assume that the size of `across` and `down` is appropriate for the given `keisuke` argument, but you should *not* make any assumptions about the order of the integers in the respective sets.


```
def columns2rows(keisuke):
    """return tuple of columns, each represented as a tuple of ints"""
    columns = []
    for column_id in range(len(keisuke[0])):
        columns.append(tuple([keisuke[i][column_id]
                             for i in range(len(keisuke))]))
    return tuple(columns)
```

[illegible]

[illegible]

Part 3: Conceptual Questions

Question 9: Algorithmic Problem Solving

[10 marks]

(a) The efficiency of algorithms is often measured in terms of “runtime efficiency” and “storage efficiency”. Briefly describe what is meant by each of these terms.

[5 marks]

(b) With the aid of an example domain or field of science, describe what is “computational simulation”.

[5 marks]

Question 10: Applications of Computing**[10 marks]**

(a) With the aid of an example, explain what is “data science”.

[5 marks]

(b) With reference to the transmission of an encrypted message, describe the role of the “private” and “public” keys in “public key cryptography”.

[5 marks]

Question 11: HTML and the Internet**[12 marks]**

(a) With reference to the following URL:

`https://www.ohim.com:443/tournament/bonus`

Identify each of the following:

[6 marks]

(i) the protocol

(ii) the host name

(iii) the port

(iv) the path

(b) Based on the following HTML document:

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
5 <title>My Web Page</title>
6 </head>
7 <body>
8 <table>
9 <tr><td></td></tr>
10 <tr><td>That's what I'm talking abo&uuml;t!</td></tr>
11 </table>
12 </body>
13 </html>
```

and the provided line numbers, identify in the document where each of the following items occurs. In the case that the item spans multiple lines, you should specify the full range of line numbers (e.g. 2–4).

[6 marks]

(i) An image

(ii) A table

(iii) A character encoding declaration

(iv) An HTML entity

[illegible]

[illegible]

[illegible]

21 of 21