Student Number

The University of Melbourne
School of Computing and Information Systems

Final Examination, Semester 1, 2019
COMP10001 Foundations of Computing

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

This paper has 37 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 3.6 (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	11	Total

Part 1: Code Interpretation

Question 1

[10 marks]

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

(a)
```
'smart array'[::-1]
```

(b)
```
7 / 2 * 2
```

(c)
```
'cat'[1:] in 'hat' and 'fox'[1:] in 'sox'
```

(d)
```
sorted({'hobbits': ['bilbo', 'frodo'],
    'dwarves': ['gimli', 'thorin']}.values())[-1][1]
```

(e)
```
[x * 2 for x in str(90210)]
```

Question 2

[9 marks]

What are the final values of each of the variables indicated below, on completion of execution of the following code:

```
VALUES = '2A'
SUITS = 'SHDC'

PLAYERS = 2
ROUNDS = 4

deck = []
for value in VALUES:
    for suit in SUITS:
        deck.append(value + suit)

hands = []
for i in range(PLAYERS):
    hands.append([])

for i in range(ROUNDS):
    for player in range(PLAYERS):
        hands[player].append(deck.pop())
```

(a) `i`

(b) `deck`

(c) `hands`

Question 3

[10 marks]

The following code is intended to calculate a list of valid “old school” vs. “new school” superhero running races. The rules are:

1. “old school” characters are defined as those who were created prior to a given year, and “new school” characters are those created in the given year or later
2. each old school superhero races against each new school superhero whose costume is a different colour to their own
3. superheroes with the same colour costumes don’t race against one another—that would just be confusing! Such pairings are deemed invalid

The code takes the form of the definition of a function called `race_list`, which takes two arguments:

- `afile`: a CSV file containing superhero data, including their name, the year they were created and their costume colour
- `year`: the year used to distinguish “old school” from “new school” superheroes

and returns a 2-tuple containing: (1) a sorted list of valid races (as strings); and (2) a count of invalid pairings of superheroes (on the basis of their costumes being the same colour).

An example input for `afile` is the file `superheroes.csv`:

```
name,year,colour
Hedgehog Man,1965,brown
Captain Penguin,1962,purple
The Mystifier,1973,purple
Telephone Girl,1978,green
Little Llama,1991,beige
```

An example function call, based on the provided `superheroes.csv` file, is:

```
>>> race_list('superheroes.csv', 1970)
(['Captain Penguin vs. Little Llama', 'Captain Penguin vs. Telephone Girl',
'Hedgehog Man vs. Little Llama', 'Hedgehog Man vs. Telephone Girl',
'Hedgehog Man vs. The Mystifier'], 1)
```

(ie, there is one invalid pairing in this example: Captain Penguin and The Mystifier do *not* race against one another because their costumes are both purple.)

You can assume that the following two library imports have been made earlier in the code, prior to the function definition:

```
import csv
import itertools
```

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), by placing line numbers in the table below the code (one line number per row in the table). Note that the provided code makes use of a semi-colon (“;”) at two different locations, to combine two statements (to initialise variables) into a single line of code.

```
1  if old[pair[0]] != new[pair[1]]:
2  old = {}; new = {}
3  with open(afile) as f:
4  new[row['name']] = row['colour']
5  else:
6  old[row['name']] = row['colour']
7  for pair in itertools.product(old, new):
8  invalid += 1
9  if int(row['year']) < year:
10 races.append(f'{pair[0]} vs. {pair[1]}')
11 for row in csv.DictReader(f):
12 else:
13 return (sorted(races), invalid)
14 def race_list(afile, year):
15 races = []; invalid = 0
```

Answer:

Question 4

[9 marks]

The following function is intended to work out the value of the optimal combination of items to place in a bag in terms of maximising the value of the items, subject to a weight limit on the bag. It takes two arguments:

- `capacity`: the total weight that can be held by the bag
- `items`: a list of items, each of which is represented as a dictionary storing the weight and value of that item

and returns the maximum value of combined items that can be achieved subject to the weight constraint.

For example, when run over the following example of `items`:

```
items = [{'weight': 13, 'value': 6}, {'weight': 15, 'value': 11},
         {'weight': 16, 'value': 13}, {'weight': 22, 'value': 17},
         {'weight': 10, 'value': 5}]
```

the function should produce the following outputs (based on the combination of items provided in the comment, in each case):

```
>>> rec_knapsack(35, items)
24   # based on items[1] and items[2]
>>> rec_knapsack(40, items)
30   # based on items[2] and items[3]
```

As presented, there are bugs in the code. Identify exactly three (3) errors in the code (using the provided line numbers), identify for each whether it is a "syntax", "run-time" or "logic" error, and provide a replacement line which corrects the error.

```
1  def rec_knapsack(capacity, items):
2      if capacity == 0 or len(items) == 0:
3          return 0
4
5      cur_item = items(0)
6
7      if cur_item['weight'] >= capacity
8          return rec_knapsack(capacity, items)
9
10     take_value = (cur_item['value']
11         + rec_knapsack(capacity - cur_item['value'], items[1:]))
12     leave_value = rec_knapsack(capacity, items[1:])
13     return max(take_value, leave_value)
```

(provide your answer on the next lined page)

Error 1:

Error 2:

Error 3:

Question 5

[16 marks]

The code on the next page is intended to validate a CSV row as containing the following four fields:

1. a **staff ID**, in the form of a 5-digit number (e.g. `00525` or `19471`)
2. a **first name**, in the form of a non-empty ASCII string
3. a **last name**, in the form of an ASCII string, noting that the field may be blank (i.e. an empty string) if the individual does not have a last name
4. a **plain text password**, in the form of an ASCII string between 8 and 12 characters in length (inclusive), including at least one lower-case letter, one upper-case letter, and one punctuation mark from: (a) a comma ("`,`"), (b) a full stop ("`.`"), (c) an exclamation mark ("!"), and (d) a question mark ("`?`").

If a valid CSV row is provided to the code (as a string input), the output of the code should be a 4-element list of the four values, each as a string; if an invalid CSV is provided to the code, the output should be `None`.

```python
def valid_name(name):
    try:
        assert name.isalpha()
        name.encode('utf-8').decode('ascii')
    except AssertionError:
        return False
    return True

def validate_row(row):
    ROW_LENGTH = 4
    STAFFID_DIGITS = 5
    MIN_PASSWORD_LEN = 8
    MAX_PASSWORD_LEN = 12
    PUNCT = ',.!? '

    fields = row.split(",")
    try:
        assert len(fields) == ROW_LENGTH
        staffid = fields.pop(0)
        assert 10**(STAFFID_DIGITS-1) <= int(staffid) < 10**STAFFID_DIGITS
        given_name = fields.pop(0)
        assert given_name and valid_name(given_name)
        last_name = fields.pop(0)
        assert valid_name(last_name)
        password = fields.pop(0)
        assert MIN_PASSWORD_LEN <= len(password) <= MAX_PASSWORD_LEN
        contains_lower = contains_upper = contains_punct = False
        for letter in password:
            if letter.islower():
                contains_lower = True
            elif letter.isupper():
                contains_upper = True
            elif not letter.strip(PUNCT):
                contains_punct = True
        assert contains_lower and contains_upper and contains_punct
    except (AssertionError, ValueError):
        return None
    return row.split(",")
```

The provided code is imperfect, in that it sometimes correctly detects valid and invalid rows, but equally, sometimes misclassifies a valid row as being invalid, and sometimes misclassifies an invalid row as being valid.

(a) Provide an example of a valid row that is correctly classified as such by the provided code (i.e. a valid row input where the return value is the row broken down into a list of valid fields):

____

(b) Provide an example of an invalid row that is correctly classified as such by the provided code (i.e. an invalid row input where the return value is `None`):

____

(c) Provide an example of an invalid row that is *in*correctly classified as a valid row by the provided code (i.e. an invalid row input where the return value is an erroneous list of valid fields):

____

(d) Provide an example of a valid row that is *in*correctly classified as an invalid row by the provided code (i.e. a valid row input where the return value is `None`):

____

Part 2: Generating Code

Question 6

[8 marks]

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

```
def n_green_bottles(n):
    while n > 0:
        b_word = 'bottles'
        if n == 1:
            b_word = b_word[:-1]
        print(f'{n} green {b_word}, sitting on the wall!')
        n -= 1
```

Question 7

[10 marks]

Write a single Python statement that generates each of the following exceptions + error messages, assuming that it is executed in isolation of any other code.

(a) `IndexError: string index out of range`

(b) `AttributeError: 'list' object has no attribute 'max'`

(c) `TypeError: unsupported operand type(s) for +: 'int' and 'list'`

(d) `ValueError: too many values to unpack (expected 2)`

(e) `TypeError: 'list' object cannot be interpreted as an integer`

Question 8 [20 marks]

“Homophily” is a sociological theory that people are more likely to be friends with those who are similar to themselves. We can quantify how homophilous a social network is by measuring the proportion of each person’s friends who share some characteristics (or “belong to the same group”) as them. Network homophily is then defined as the average of these proportions over all people in the network.

The code provided below calculates the network homophily of a network `net` with respect to `groups`. A network is specified as a dictionary of sets of friends of each person. Groups are specified as a list of sets of people belonging to the same group. The function returns the proportion of a person’s friends who belong to the same group as them, averaged over all people in the network.

You can assume that `net` contains at least two people, that each person has at least one friend, that all people contained in `groups` are found in `net`, and that each person belongs to exactly one group.

For example, given `net` and `groups` initialised as follows:

```
net = {'ada': {'bo', 'dave', 'eiji'},
       'bo': {'ada', 'carla', 'dave'},
       'carla': {'bo', 'dave'},
       'dave': {'ada', 'bo', 'carla', 'eiji'},
       'eiji': {'ada', 'dave'}}

groups = [{'ada', 'bo', 'carla'},
          {'dave', 'eiji'}]
```

The code should function as follows:

```
>>> network_homophily(net, groups)
0.45
```

Complete the code by providing a single statement to insert into each of the numbered boxes. Note that your code should run at the indentation level indicated for each box.

```
def network_homophily(net, groups):
    [         1          ]
    group_id = 0
    while group_id < len(groups):
        [         2          ]
            membership[member] = group_id
        group_id += 1

    proportions = []
    for person in net:
        same_count = 0
        [         3          ]
        for neighbour in neighbours:
            [         4          ]
                same_count += 1
        proportions.append(same_count / len(neighbours))

    [         5          ]
    return average_proportion
```

(1) ______________________________

(2) ______________________________

(3) ______________________________

(4) ______________________________

(5) ______________________________

Part 3: Conceptual Questions and Applications of Computing

Question 9: Computing Fundamentals

(a) One of the desiderata for an algorithm is that it should be "correct". Describe in one sentence what is meant by an "incorrect" algorithm.

[2 marks]

(b) Is the following statement True or False; include a justification as part of your answer:

A text document will always be smaller when encoded in UTF-8 than UTF-16.

[4 marks]

(c) What does it mean for an algorithm or method to be *dual use*? Illustrate with the use of an example.

[3 marks]

Question 10: Applications of Computing

(a) What are three issues that must be addressed in an “electronic voting” system?

[6 marks]

(b) How is the state of a “quantum bit” (or “qubit”) defined?

[3 marks]

Question 11: URLs and the Web [10 marks]

Complete this HTML page:

```
<![      1      ] html>
<[      2      ] xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
<[      3      ] http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
<title>COMP10001: The Final Exam</title>
</head>
<body>
<h1>COMP10001: The Final Exam</h1>
<p>Starring:
<ul>
  <li>You</li>
  <li>Tim, Nic, Marion, Farah</li>
  <li><[      4      ] src="./images/totoro.gif" alt="Totoro!"/></li>
</ul>
</p>
</[      5      ]>
</html>
```

by filling in the numbered blanks based on selecting from among the following candidate strings:

```
a          body       DOCTYPE    href       html
img        meta       ol         src        td
```

Note that you may use the same string for multiple answers.

(1) ______________________________

(2) ______________________________

(3) ______________________________

(4) ______________________________

(5) ______________________________

⇜⋐⋐⋐⋐⋐⋐⋐ End of Exam ⋑⋑⋑⋑⋑⋑⋑⇝