COMP10001 Foundations of Computing
Final Exam Preparation

Semester 1, 2019

Tim Baldwin, Nic Geard, Farah Khan, and Marion Zalk

— version: 1597, date: May 30, 2019 —

Announcements

- Don't forget to provide subject feedback via the SES:
  `https://ses.unimelb.edu.au`

Lecture Agenda

- Last lecture:
  - Final exam preparation
- This lecture:
  - Final exam preparation (cont.)

Lecture Outline

1. Practice Exam: Part 1 (cont.)
2. Practice Exam: Part 2
3. Practice Exam: Part 3

Other Common Part 1 Question Types

- Another common question type relates to code with logic errors, and the identification of test cases of the following four classes:
  1. Provide an example of an input which is correctly classified as `True` (a "true positive")
  2. Provide an example of an input which is correctly classified as `False` (a "true negative")
  3. Provide an example of an input which is incorrectly classified as `True` (a "false positive")
  4. Provide an example of an input which is incorrectly classified as `False` (a "false negative")

Lecture Outline

1. Practice Exam: Part 1 (cont.)
2. Practice Exam: Part 2
3. Practice Exam: Part 3

Question 4

```python
1  def get_message(filename, which):
2      text = open("filename").read()
3      lines = text.split(',')
4      message = ''
5      for line in lines:
6          if which == 1:
7              index = len(line)
8          else:
9              index = 0
10         message += line(index)
11     return message
```

Identify exactly three (3) errors and specify: (a) the line number where the error occurs; (b) the type of error; and (c) how you would fix each error.

Question 4 (cont.)

A:

- line 2 (run-time [if no file of name "filename"] OR logic): `text = open(filename).read()`
- line 3 (logic): `lines = text.split('\n')`
- line 7 (run-time [on line 10]): `index = len(line) - 1`
- line 10 (run-time): `message += line[index]`
- line 11 (logic): `return message.upper()`

Question 5

```python
import csv

def csv_aggregate(infile, outfile):
    [ 1 ]
    fpout.write("docid,pos_ratio\n")
    prev_docid = None
    [ 2 ]
    for row in csv.DictReader(open(infile)):
        [ 3 ]
            if prev_docid != None:
                fpout.write(f"{prev_docid},{pos/total}\n")
            [ 4 ]
            pos = total = 0
```


Question 5 (cont.)

```python
import csv

def csv_aggregate(infile, outfile):
    fpout = open(outfile, "w")
    fpout.write("docid,pos_ratio\n")
    prev_docid = None
    [ 2 ]
    for row in csv.DictReader(open(infile)):
        [ 3 ]
            if prev_docid != None:
                fpout.write(f"{prev_docid},{pos/total}\n")
            [ 4 ]
            pos = total = 0
```


Question 5 (cont.)

```python
import csv

def csv_aggregate(infile, outfile):
    fpout = open(outfile, "w")
    fpout.write("docid,pos_ratio\n")
    prev_docid = None
    pos = total = 0
    for row in csv.DictReader(open(infile)):
        [ 3 ]
            if prev_docid != None:
                fpout.write(f"{prev_docid},{pos/total}\n")
            [ 4 ]
            pos = total = 0
```


Question 5 (cont.)

```python
import csv

def csv_aggregate(infile, outfile):
    fpout = open(outfile, "w")
    fpout.write("docid,pos_ratio\n")
    prev_docid = None
    pos = total = 0
    for row in csv.DictReader(open(infile)):
        if row["docid"] != prev_docid:
            if prev_docid != None:
                fpout.write(f"{prev_docid},{pos/total}\n")
            [ 4 ]
            pos = total = 0
```

Question 5 (cont.)

```
import csv

def csv_aggregate(infile, outfile):
    fpout = open(outfile, "w")
    fpout.write("docid,pos_ratio\n")
    prev_docid = None
    pos = total = 0
    for row in csv.DictReader(open(infile)):
        if row["docid"] != prev_docid:
            if prev_docid != None:
                fpout.write(f"{prev_docid},{pos/total}\n")
            prev_docid = row["docid"]
            pos = total = 0
```


Question 5 (cont.)

```
        if row["pos"] == '1':
            pos += 1
        total += 1
    if prev_docid != None:
        [          5          ]
    fpout.close()
```


Question 5 (cont.)

```
        if row["pos"] == '1':
            pos += 1
        total += 1
    if prev_docid != None:
        fpout.write(f"{prev_docid},{pos/total}\n")
    fpout.close()
```


Question 6

Write a function `equiword(word)` that takes a single argument `word` (a non-empty string) and returns a (positive) integer `n` if all unique letters in `word` occur exactly `n` times, and `False` otherwise.

Question 6 (cont.)

```
from collections import defaultdict

def equiword(word):
    lcount = defaultdict(int)
    for letter in word:
        lcount[letter] += 1
    values = list(set(lcount.values()))
    if len(values) == 1:
        return values[0]
    return False
```


Other Common Part 2 Question Types

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

```
def last_letter(word):
    last = 0
    for i in range(1, len(word)):
        if word[i] > word[last]:
            last = i
    return last
```

Other Common Part 2 Question Types

Answer:

```
def last_letter(word):
    last = 0
    i = 1
    while i < len(word):
        if word[i] > word[last]:
            last = i
        i += 1
    return last
```


Other Common Part 2 Question Types

- 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, e.g.:
  - `TypeError: unsupported operand type(s) for +: 'int' and 'str'`
  - `ValueError: invalid literal for int() with base 10: 'a'`

Marking Part 2 Code

- Our approach in marking Part 2 code is as follows:
  - syntax error < run-time error < logic error
  - don't multiply penalise reoccurrences of the same syntax error within a given question
  - if error found, continue marking subsequent code as if that error had been debugged (i.e. don't cascade errors in marking)
  - no direct marking of efficiency or redundancy, but your code must generalise (not just work for the supplied examples)

Lecture Outline

1. Practice Exam: Part 1 (cont.)
2. Practice Exam: Part 2
3. Practice Exam: Part 3

Question 7a

What is an "algorithm" in the context of Computing?

Question 7a

What is an "algorithm" in the context of Computing?

A: *An algorithm is a set of steps for solving an instance of a particular problem type*

Question 7b

Outline the "generate-and-test" strategy of algorithmic problem solving. With the aid of an example, explain what sort of problems it is commonly applied to.

COMP10001 Foundations of Computing Week 12, Lecture 2 (30/5/2019)

Question 7b

Outline the "generate-and-test" strategy of algorithmic problem solving. With the aid of an example, explain what sort of problems it is commonly applied to.

A: *Generate candidate answers and test them one by one until a solution is found; assumes a candidate answer is easy to test and that the set of candidate answers is ordered or can be generated exhaustively. It is commonly applied to problems where the number of solutions is relatively small, such as small-scale combinatoric problems (e.g. logic puzzles).*