COMP10001 Foundations of Computing
Final Exam Preparation

Summer Term, 2021
Ekaterina Vylomova

THE UNIVERSITY OF MELBOURNE

— VERSION: —-, DATE: FEBRUARY 18, 2021 —

Lecture Agenda

- Last lecture:
  - Counting, Encodings, Text Representation
- This lecture:
  - Ethics and Bias
  - Final exam preparation

Lecture Outline

1. Ethics and Bias
2. Preparing for the Final Exam
3. Practice Exam
4. Practice Exam: Part 2
5. Practice Exam: Part 3
6. SPOILER ALERT: Practice Exam with Solutions
7. Practice Exam: Part 2 with Solutions
8. Practice Exam: Part 3 with Solutions

Ethics of Computing

- Why ethics all of a sudden?
  - computing can trivially be deployed at a global scale, and has real potential to change the world
  - computing can trivially be applied in all sorts of contexts over all sorts of data ... not everything that *can* be computed *should* be computed
- Examples of unethical applications that you could relatively easily implement based on your COMP10001 knowledge (with a bit of upskilling with the right libraries):
  - spam generators
  - DDoS attack systems to bring down web services
  - web crawlers to indiscriminately crawl the web

ACM Code of Ethics and Professional Conduct

1. Contribute to society and to human well-being, acknowledging that all people are stakeholders in computing
2. Avoid harm
3. Be honest and trustworthy
4. Be fair and take action not to discriminate
5. Respect the work required to produce new ideas, inventions, creative works, and computing artefacts
6. Respect privacy
7. Honor confidentiality

Source(s): `https://www.acm.org/code-of-ethics`

Ethical Data Usage: Cambridge Analytica

- Cambridge Analytica:
  - Facebook app released, which crawled all the data of the individual, and all of their friends
  - 270k users used the app, generating a dataset of 87m Facebook users
  - that data was then used to place targeted ads, with the explicit intent of manipulating voter behaviour in the US presidential elections
- What terms of the ACM Code of Ethics did it violate?

Dual Use

- In the context of computing, "dual use" refers to technologies that can equally be used for good or malicious purposes
- Examples from AI:
  - autonomous cars
  - face recognition
  - machine translation
  - video generation ("deep fake")

Ethical Data Use

- It is trivially easy to collect data, but the governance/use of any dataset should always consider (at least) the following issues:
  - **privacy**: who owns the data, what access do others have to it, and what does it expose about users both directly and indirectly?
  - **governance**: management of the availability, integrity, security, etc. of the data
  - **fairness**: respect the people behind the data, and ensure that all benefit equally
  - **shared benefit**: those who produce the data should benefit from it
  - **transparency**: be clear as to how the data is used, by whom, and for what

Bias and Fairness

- Computing and AI is heavily data driven, meaning it is highly susceptible to data bias
- AI systems often not only capture data bias, but tend to amplify it:
  - image labelling
  - speech recognition
  - medical diagnosis
- Important to both quantify and mitigate bias to achieve fairness

Bias and Fairness

- So what?

  critically important that AI doesn't favour a particular demographic in society, or exacerbate stereotypes/reinforce stigmas ... for this we need to develop more sophisticated understanding of our datasets, training algorithms, and better methods for detecting/quantifying such biases

Section Summary

- What are the general principles of the ACM Code of Ethics?
- What is dual use in the context of computing?
- What are the terms of ethical data use?

Lecture Outline

1. Ethics and Bias
2. Preparing for the Final Exam
3. Practice Exam
4. Practice Exam: Part 2
5. Practice Exam: Part 3
6. SPOILER ALERT: Practice Exam with Solutions
7. Practice Exam: Part 2 with Solutions
8. Practice Exam: Part 3 with Solutions

Tips for Exams

- Like the MST, the Final Exam will be hosted on Grok, and accessed via the LMS
- Like the MST, the Final Exam will be open book, but you must not communicate with anyone else during the exam (it must be your own, independent, original work)

Tips for Exams

- It is your responsibility to make sure you know the time and date of the Final Exam, and to monitor Announcements on the COMP10001 LMS site.
- Lecturers do not have the authority to grant special consideration if you miss the exam.
- While we have tried to be as accommodating as possible if difficulties occurred with other assessment, we don't have that flexibility with the Final Exam.

Tips for Exams

- If you were impeded by illness or other documentable causes, apply for special consideration *within 48 hours of the exam via Stop1*, (**not** the lecturers)
- Use the past exams as part of your preparation, but note that pre-2015 exams are based on **Python2** (and pre-2018 exams are not Python 3.6), and pre-2020 were on paper (not Grok). We will post an Announcement when past papers are on the LMS.
- The exam focus varies from semester to semester, based on the focus of that particular subject instance

Tips for Exams

- Practice for the exam by doing old exam questions, tutorial questions, Grok worksheet exercises
- Start with a blank sheet of paper, and write out your code (don't look at the answers)
- Type your answer into Grok (or Python Tutor) and test it out
- Make a list of the common mistakes that you make, and write a remidner to yourself on how to avoid them
- Then look at the solutions, and see how they worked

What's Examinable?

- Everything from all the lectures/tutorials
- We won't ask questions that require you to **write** code (from scratch) that uses recursion or comprehensions
- We won't ask you to **write** code to generate anything other than the most basic HTML document (e.g. simple list or table)

Assessment

- Your subject mark will be made up of:
  - Interactive Grok Learning worksheets: 10%
  - Projects ($\times 2$): 30%
  - Mid-semester test: 10%
  - Final exam: 50%

Assessment

- There are two "hurdles" for the subject: you must achieve at least 50% for the projects/interactive worksheets
  AND at least 50% for the mid-semester test/final exam
  Of course, you must also achieve at least 50% overall
  (at least 50 out of 100 possible marks)

Lecture Outline

1. Ethics and Bias
2. Preparing for the Final Exam
3. Practice Exam
4. Practice Exam: Part 2
5. Practice Exam: Part 3
6. SPOILER ALERT: Practice Exam with Solutions
7. Practice Exam: Part 2 with Solutions
8. Practice Exam: Part 3 with Solutions

Question 1

Construct a **single** Python expression which evaluates to the following values, and incorporates the specified operations in each case (executed in any order).

Tips for this Style of Question

- Your code must be in a SINGLE statement (not over multiple lines; no semi-colons; ...)
- No `print` functions
- The order of the operators doesn't matter, but they should all be there
- Make sure the output matches the value + type indicated
- Use the code buffer to test your code

Question 2

The following function is intended to ...
As presented, the lines of the function, shown below, are out of order. Put the lines in the correct order and introduce appropriate indentation.

Tips for this Style of Question

- Make sure to read the description carefully before diving in
- Expect more lines (and a bit more complexity) than in the MST
- Look out for dependencies between lines (variable assignments; `if` vs. `else`; ...)
- The code will deliberately be made so that you can't run it (directly)

Question 3

- Given code, identify of test cases of the following four classes:
  1. Provide an example of a valid input which is correctly classified as `True` (a "true positive")
  2. Provide an example of an invalid input which is correctly classified as `False` (a "true negative")
  3. Provide an example of an invalid input which is incorrectly classified as `True` (a "false positive")
  4. Provide an example of a valid input which is incorrectly classified as `False` (a "false negative")

Other Common Part 1 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:
- Note that unnecessary or redundant statements in your answer may be penalised in these types of `for`/`while` questions

Other Common Part 1 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'`

Lecture Outline

1. Ethics and Bias
2. Preparing for the Final Exam
3. Practice Exam
4. Practice Exam: Part 2
5. Practice Exam: Part 3
6. SPOILER ALERT: Practice Exam with Solutions
7. Practice Exam: Part 2 with Solutions
8. Practice Exam: Part 3 with Solutions

Question 4

Given some code ...
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 5

Given some code with some blank sections ...
Fill in the blanks.

Question 6

Write a function XYZ that takes an argument ...

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. Ethics and Bias
2. Preparing for the Final Exam
3. Practice Exam
4. Practice Exam: Part 2
5. Practice Exam: Part 3
6. SPOILER ALERT: Practice Exam with Solutions
7. Practice Exam: Part 2 with Solutions
8. Practice Exam: Part 3 with Solutions

Question 7

- Conceptual questions: With the aid of an example, explain ...
- If your answer is more than 2 or 3 sentences,
  it is probably too long
- Irrelevant information may be penalised

How We Ensnared You ...

- Harnessing computation for problem solving
- Fundamental programming constructs
- Data manipulation
- The Web, multimedia and visualisation
- Elements of maths, engineering, logic, design; dollops of creativity
- Concerned with theories, principles, limits of computation and information
- If you enjoy puzzles, argument, philosophy and games ... oh and *fun*, you've come to the right place!

Onwards and Upwards ...

- Computing and Software Systems major (BSc)
  - The next stop is ... COMP10002 Foundations of Algorithms
  - And perhaps also ... INFO20003 Database Systems, COMP20008 Elements of Data Processing, and INFO10003 Fundamentals of Interaction Design

Onwards and Upwards ...

- Computing and Software Systems major (BSc)
- Data Science major (BSc)
  - The next stop is ... COMP10002 Foundations of Algorithms
  - And perhaps also ... INFO20003 Database Systems, COMP20008 Elements of Data Processing, and INFO10003 Fundamentals of Interaction Design

Onwards and Upwards ...

- Computing and Software Systems major (BSc)
- Data Science major (BSc)
- Diploma of Computing

Grok Tutoring Help

- With the term finishing this week, we will no longer be supporting the Grok Tutoring Help facility beyond this Friday, and tutors will no longer be monitoring the forums
- We will, however, run a consultation session closer to the exam; watch the LMS for details

Final Words

- Thanks for giving computing a go! The COMP10001 team:

  Akira
  Hummad

  Lucien

  Nathen

  Xiu Meng

  Andreas

  the Groksters

  Kat

Lecture Outline

1. Ethics and Bias
2. Preparing for the Final Exam
3. Practice Exam
4. Practice Exam: Part 2
5. Practice Exam: Part 3
6. SPOILER ALERT: Practice Exam with Solutions
7. Practice Exam: Part 2 with Solutions
8. Practice Exam: Part 3 with Solutions

Question 1a

Construct a **single** Python expression which evaluates to the following values, and incorporates the specified operations in each case (executed in any order).

- Output value: `'grin'`
- Required operations:
  - string slicing
  - string indexing

Question 1a

Construct a **single** Python expression which evaluates to the following values, and incorporates the specified operations in each case (executed in any order).

- Output value: `'grin'`
- Required operations:
  - string slicing
  - string indexing

A:

```
'gristle'[:3] + 'nincompoop'[0]
```

Question 1b

- Output value: `True`
- Required operations:
  - `range` function
  - `in`

Question 1b

- Output value: `True`
- Required operations:
  - `range` function
  - `in`

A:

```
2 in range(3)
```

Question 1c

- Output value: `2`
- Required operations:
  - (float) division
  - integer division

Question 1c

- Output value: `2`
- Required operations:
  - (float) division
  - integer division

A:

```
int(8/2//2)
```

Question 1d

- Output value: `0`
- Required operations:
  - dictionary lookup
  - `pop` method
  - list indexing

Question 1d

- Output value: `0`
- Required operations:
  - dictionary lookup
  - pop method
  - list indexing

A:

```
{'success': [['excitement', 11], ['concern', 0]]}
    ['success'].pop()[-1]
```

Question 1e

- Output value: `'ace'`
- Required operations:
  - `items` method
  - list indexing
  - tuple indexing

Question 1e

- Output value: `'ace'`
- Required operations:
  - `items` method
  - list indexing
  - tuple indexing

A:

```
sorted({(1,1): "racehorse", (2,2):
    "one too"}.items())[0][1][1:4]
```

Tips for this Style of Question

- Your code must be in a SINGLE statement (not over multiple lines; no semi-colons; ...)
- No `print` functions
- The order of the operators doesn't matter, but they should all be there
- Make sure the output matches the value + type indicated
- Use the code buffer to test your code

Question 2

The following function is intended to calculate the similarity between two names by calculating what proportion of "*n*-grams" (substrings of *n* characters) from the first name can be found in the second name, and averaging across the resulting values for each value of *n* (by default, the values 2 and 3). If the first name has fewer characters than any tested value of *n*, the function should return 0.0.

Question 2

```
def name_similarity(name1, name2, maxn=4):
    sim_list = []
    for n in range(2, maxn):
        if len(name1) >= n:
            matches = 0
            for start in range(0, len(name1) - (n - 1)):
                ngram = name1[start:start + n]
                if ngram in name2:
                    matches += 1
            sim_list.append(matches/(len(name1) - (n - 1)))
    if sim_list:
        return sum(sim_list)/len(sim_list)
    return 0.0
```

Tips for this Style of Question

- Make sure to read the description carefully before diving in
- Expect more lines (and a bit more complexity) than in the MST
- Look out for dependencies between lines (variable assignments; `if` vs. `else`; ...)
- The code will deliberately be made so that you can't run it (directly)

Question 3

- Given code, identify of test cases of the following four classes:
  1. Provide an example of a valid input which is correctly classified as `True` (a "true positive")
  2. Provide an example of an invalid input which is correctly classified as `False` (a "true negative")
  3. Provide an example of an invalid input which is incorrectly classified as `True` (a "false positive")
  4. Provide an example of a valid input which is incorrectly classified as `False` (a "false negative")

Question 3: Sample Answers

a. `'12345,Kim,Person,passWORD?'`
b. `'XX,,Person,'`
c. `'12345,Kim,Person,pass WORD?'`
d. `'01234,Kim,,passWORD,'`

Other Common Part 1 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 1 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 1 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'`

Lecture Outline

1. Ethics and Bias
2. Preparing for the Final Exam
3. Practice Exam
4. Practice Exam: Part 2
5. Practice Exam: Part 3
6. SPOILER ALERT: Practice Exam with Solutions
7. Practice Exam: Part 2 with Solutions
8. Practice Exam: Part 3 with Solutions

Question 4

```
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

```
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.)

```
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.)

```
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.)

```
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
```

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. Ethics and Bias
2. Preparing for the Final Exam
3. Practice Exam
4. Practice Exam: Part 2
5. Practice Exam: Part 3
6. SPOILER ALERT: Practice Exam with Solutions
7. Practice Exam: Part 2 with Solutions
8. Practice Exam: Part 3 with Solutions

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.

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).*