Online Tutoring to Support Programming Exercises

Bryn Jeffries bryn@groklearning.com
Ben Taylor ben@groklearning.com

Tim Baldwin tbaldwin@unimelb.edu.au
Marion Zalk mzalk@unimelb.edu.au
Rationale for online tutoring

**Lectures & Classes**
- ✗ Unsuitable for resolving individual students’ issues
- ✗ Out of sync with student activity

**Individual tutoring**
- ✓ Tailored response to student questions
- ✗ Expensive to scale
- ✗ Hard to schedule

**Course forums**
- ✓ No time constraints
- ✓ Scales to large groups
- ✓ Allows peer support
- ✗ Moderation issues
- ✗ Reluctance to share or expose misunderstandings to group
- ✗ Poorly suited to individual student issues
- ✗ Can’t discuss assessable code
Problem description

We begin with a feature extraction function. The features we are going to use are called trigrams. A trigram is simply a string of three contiguous characters. For example in the string "I love computing", there are lots of trigrams (L – 2 to be precise, where L is the length of the string): "I l", "l o", "o ve", "ve m", "m e", "e n" are the first four of them, in sequence.

Write a function `count_trigrams(document)` that takes a string and returns a dictionary with the frequency counts of the trigrams within the string (noting that if you have N repeats of the same trigram in the string, the frequency will be N). Note that the output must be a default dictionary and not a standard dictionary, as it will be useful later. Note also that you should not modify the string in any way (e.g. remove punctuation, remove whitespace or convert to lower case) in calculating the frequencies.

Your code should behave as follows:

```python
>>> count_trigrams("hello")
defaultdict(<class 'float'>, {'hel': 1.0})
>>> count_trigrams("aaaaa")
defaultdict(<class 'float'>, {'aaa': 3.0})
>>> count_trigrams("Boaty McBoatFace")
defaultdict(<class 'float'>, {'Boa': 2.0, 'oat': 2.0, 'face': 1.0})
```
Student view: Student has made an attempt at the problem, and is about to run their code.
Student view: Running the code causes a TypeError, and the student does not know how to proceed.
Student view: Student opens tutor chat window and requests help.
Triage page: Tutors monitor all students’ request, and from here can choose to open specific requests.
Tutor view: Tutor can see the student’s code history, and try out changes in a sandbox that doesn’t affect the student’s own code. The tutor then makes a suggestion to the student in the chat window.

Reminder: The greater than symbol is >

3. Add an else to the end. Inside the else statement, print out No record yet. Keep training!
Student view: Student has continued to attempt questions, and is alerted that a tutor has responded to their request.
Student view: Student checks the tutor’s response in the problem’s chat window. The student could ask further questions in this thread, but in this case they have enough to proceed.
Write a program that asks how many tricks your cat can do. If the number is greater than 24 then it should do this:

```
How many tricks? 28
New world record!
```

Otherwise it should do this:

```
How many tricks? 5
No record yet. Keep training!
```

1. Ask for the number of tricks with `input` and convert it to a number with `int`.
2. Add an `if` to check if the number of tricks is greater than 24. If it is greater than 24 then print out New world record!
   *Reminder: The greater than symbol is >*
3. Add an `else` to the end. Inside the `else` statement, print out No record yet. Keep training!

Run your program and check it with the first example: 28. Then check it with the second example: 5.
Student view: Student submits their code for marking, and is informed that one of the tests did not pass.
Student view: This time the student has enough information to identify the bug in their code (missing whitespace), and applies a fix without further interaction with a tutor.
Student view: The student submits their code for marking again, and this time passes all the tests.
Schools Course: NCSS Challenge (2018)

- Blockly
- Python

12,734

5 weeks

5* problems per week

170

2-3 on duty

8am—9pm

7 days/week

Alert on 5 failed submissions
University Subject: COMP10001 Foundations of Computing (2018)

- Python
- 1,151 (S1)
- 641 (S2)
- 12 weeks
- 2—9 worksheet problems per week (10%)
- 23 (S1)
- 19 (S2)
- 1—2pm, 7—8pm weekdays
- Forum for non-code questions
Chat Usage Times: COMP10001

Monitored period

Live tutor hours
Canned Responses

General problem: Hidden Tests/Debug Strategies

Hi there, we aren't able to comment on the contents of hidden inputs, but to point you in the right direction you'll want to check the specifications carefully and ensure your program conforms to it. Make sure you're clear on the details of the specifications and the details of how your program works - as you'll need to know both to ensure your program is behaving correctly.

Note: You do not need to *guess* what hidden inputs are! If you've implemented the specifications correctly, then your program will pass the hidden inputs.

Have a think about the place(s) that a logical error could occur in your code, and...

Common concept: Sorting using a custom key

If you want to sort a list where each element is not just one thing, but a collection of things like a list of a tuple then you can sort the list of lists or list of tuples by using the `sorted()` function with some extra arguments.

We want to sort, but sort based off a key that we choose. Originally, the `sorted()` function sorts in ascending order, number, for example `1 < 2` and `5 < 7`.

Specific problem: Building a string for the answer

To solve this question, you can't have a `print` statement inside your `for` loop. Instead, you'll need to use another string variable to build up the final answer one character at a time.

Then, once the `for` loop is finished, you can print out the final answer string.
Overly dependent students
Problem Outcome Modelling: Discrete-Time Markov Chains

NCSS Challenge exc. Advanced
Summary

Online tutoring
• Is personalised
• Connects with humans
• Is private
• Complements auto-grading feedback
• Untethered from the classroom
• Works on scale
• Can be incorporated into university teaching budget
• Does not need to be monitored 24/7

Benefits from
• Training of tutors to guide students (and deter dependency)
• Use of canned responses where appropriate
• Spotting students in need (on path of despair)
• Promoting awareness