COMP10001 Foundations of Computing
Semester 1, 2019

Tutorial Questions: Week 8

— VERSION: 1474, DATE: APRIL 10, 2019 —

Discussion

1. What is a "bug"? What are some debugging strategies?

   A: *In computing, a bug is an error in code which causes a program to not run as intended. Debugging strategies include running tests to isolate cases where the problem occurs, and using diagnostic print statements in parts of your code to see what the value of variables are during execution.*

2. What are the three types of errors we've learned about and what do they mean?

   A: *The three errors are (1) Syntax error: where the code won't run due to incorrect use of symbols in the programming language; (2) run-time error: where the code runs but encounters a problem during execution causing the program to crash; and (3) logic error: where the code runs to completion without problem, but the result is not what the developer intended.*

   Now try Exercises 1 & 2

3. What is "lazy evaluation"? How can it make our code safer and more efficient?

   A: *Lazy evaluation is the technique where, in certain circumstances, an answer is returned as soon as it is known. It's often the case where a sequence of values is iterated through and a test applied to each one. If we're looking to find whether one value fulfils the test, we can return a* `True` *result as soon as we find a single such value: there's no need to continue testing the rest of the values as nothing will change the output from being* `True`*.*

4. What is "`None`"? How is it used?

   A: `None` *is a special value in python, notable for being what's passed as the return value of a function when no return value is specified. It can therefore be used to represent the absence of a result, perhaps as a somewhat third option to a True/False boolean result.* `None` *is its own type, so nothing else has equality with it.*

   Now try Exercise 3

Exercises

1. Imagine we are testing a function which has been written. It's supposed to take an employee's ID and a year as input, fetch their records and calculate the total amount of tax which they paid in that year. Describe five aspects of the code's functionality you would test if you were to write test cases.

   A: *Some cases include: an employee ID which doesn't exist; a year in the past where the employee was not yet hired and so has no records; an employee who no longer works but should have records for a previous year in the system; the current year (in which case a partial answer would be given); a year where they perhaps earned below a tax-free threshold son paid no tax; and of course a well formed input which should return a correct result.*

2. Find the errors in the following programs. Classify them as (a) syntax, (b) runtime or (c) logic errors.

   (a)
   ```
   def disemvowel(text):
       """ Returns string `text` with all vowels removed """
       vowels = ('a', 'e', 'i', 'o', 'u')
       answer = text[0]
       for char in text:
           if char is not in vowels:
               answer = char + answer
       print(answer)
   ```

   A: *Below is the line number of the error, the type of error and a replacement line of code to fix it.*

   - *line 4; logic/run-time (if empty string);* `answer = ''`
   - *line 6; syntax;* `if char not in vowels:`
   - *line 7; logic;* `answer += char`
   - *line 8; logic;* `return answer`

(b)
```
def big-ratio(nums, n):
    """ Calculates and returns the ratio of numbers
    in list `nums` which are larger than `n` """
    n = 0
    greater_n = 0
    for number in nums:
        if number > n:
            greater_n += 1
            total += 1
    return greater_n / total

nums = [4, 5, 6]
    low = 4
print(f"{100*big_ratio(nums, low)}% of numbers are greater than {low}")
```

A:

- *line 1; syntax;* `def big_ratio(numlist,n):`
- *line 4; logic;* `total = 0`
- *line 9; logic; correct indentation for total (outside* `if` *block)*
- *line 13; syntax; remove indentation*

3. Compare the two functions below. Are they equivalent? Why would we prefer one over the other?

```
def noletter_1(words, letter='z'):
    for word in words:
        if letter in word:
            return False
    return True

def noletter_2(words, letter='z'):
    no_z = True
    for word in words:
        if letter in word:
            no_z = False
    return no_z

wordlist = ['zizzer'] + ['aardvark'] * 1000000
print(noletter_1(wordlist))
print(noletter_2(wordlist))
```

A: *The two functions are functionally equivalent, but the first one uses lazy evaluation while the second one doesn't. In the example of* `'zizzer'` *followed by a million instances of* `'aardvark'`*, the first function will perform much faster as it's able to return* `False` *as soon as it tests* `'zizzer'`*. The second will continue to iterate through every instance of* `'aardvark'` *before returning* `False`*, taking much more time unnecessarily.*

Problems

1. Write a program which asks the user to enter a series of words and then checks whether any of those words are palindromes (spelled the same way backwards as forwards, like hannah). You should print True if there are any palindromes and False if there are none. Use lazy evaluation to save some time

A:

```
def palindromes(words):
    for word in words:
        if word == word[::-1]:
            return True
    return False

words = input("Enter words: ")
print(palindromes(words.split()))
```

2. Write a function which takes a string, a character and an integer threshold and returns `True` if the character appears in the string with a frequency above the threshold, `False` if it appears below the threshold and `None` if it doesn't appear at all.

A:

```
def above_thresh(text, char, thresh):
    if char not in text:
        return None
    freq_dict = {}
    for letter in text:
        if letter in freq_dict:
            freq_dict[letter] += 1
        else:
            freq_dict[letter] = 1
    return freq_dict[char] > thresh
```