COMP10001 Foundations of Computing
Semester 1, 2022

Tutorial Questions: Week 8

— VERSION: 1479, DATE: APRIL 26, 2022 —

Discussion

1. What is a "library"? How do we access them?
2. What is a "`defaultdict`"? How do we initialise and use it?

   Now try Exercise 1
3. What is a "bug"? What are some debugging strategies which we can use when we find an error?
4. What are the three types of errors we've learned about?
5. How can we use testing to find bugs or confirm our code runs properly? What are some strategies we can adopt to write comprehensive test cases for our code?

   Now try Exercises 2 – 4

Exercises

1. Rewrite the following with a default dictionary

```
my_dict = {}
for i in range(10):
    if i % 3 in my_dict:
        my_dict[i % 3].append(i)
    else:
        my_dict[i % 3] = [i]
```

2. Find the errors in the following programs, classifying them as (a) syntax, (b) runtime or (c) logic errors. Fix them with a correct line of code.

(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.lower() is not in vowels:
            answer = char + answer
    print(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}")
```

3. The function below is supposed to take a list of integers and remove the negative integers from the list, however, it is not working as intended.

   - Write down three test cases that could be useful for function verification or finding bugs.
   - Debug the associated code snippet to solve the problem.

```
def remove_negative(nums):
    for num in nums:
        if num < 0:
            nums.remove(num)
```

4. This question is based on a previous exam question. The code below is intended to validate a data entry with the following three fields:

   (a) a *staff ID*, in the form of a string containing a 5-digit number (e.g. `"00520"` or `"19471"`)

   (b) a *first name*, in the form of a non-empty string of alphabetical letters

   (c) a *password*, in the form of a string including at least one lower-case letter, one upper-case letter, and one punctuation mark from the following `[',', '.', '!', '?']`

   The function should return `True` if the data entry contains entirely valid values (according to the above rules) and `False` if any of the fields are invalid.

```
STAFFID_LEN = 5

def validate(data):

    staffid = data.pop(0)
    if not 10**(STAFFID_LEN-1) <= int(staffid) < 10**STAFFID_LEN:
        return False

    first_name = data.pop(0)
    if not first_name and first_name.isalpha():
        return False

    password = data.pop(0)
    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(',.!?␣'):
            contains_punct = True

    if not contains_lower and contains_upper and contains_punct:
        return False

    return True
```

   The provided code is imperfect, in that sometimes it validates and invalidates correctly, but sometimes it misclassifies valid data as invalid and invalid data as valid. Your task is to:

   (a) Provide an example of valid data that is correctly classified as such by the provided code (i.e. valid data input where the return value is `True`).

   (b) Provide an example of invalid data that is correctly classified as such by the provided code (i.e. invalid data input where the return value is `False`).

   (c) Provide an example of invalid data that is *in*correctly classified as a valid by the provided code (i.e. valid data input where the return value is erroneously `True`).

   (d) Provide an example of valid data that is *in*correctly classified as an invalid by the provided code (i.e. invalid data input where the return value is erroneously `False`).

Problems

1. Write a function which takes a string as input and returns a sorted list of the words which occur only once in the string. Try using a defaultdict and list comprehensions in your solution.
   `once_words('the␣cat␣in␣the␣hat␣is␣a␣cat')` should return `['a', 'hat', 'in', 'is']`.

2. Write a function which takes a string containing an FM radio frequency and returns whether it is a valid frequency. A valid frequency is within the range 88.0-108.0 inclusive with 0.1 increments, meaning it must have only one decimal place. `valid_fm('103.14')` should return `False`.