

# algorithm

Terry Yin

October 21, 2014

I've done all the 3 options (I posted only one). Hopefully that won't give me any negative points as I'm not following the instruction technically. I'm a big fan of programming exercises. Just cannot stop.


This paper is written in iPython Notebook [Pérez & Granger \(2007\)](#).

## 1 Option 1: Numbers

Your algorithm should:


- Search a string of at least five numbers (for example, 37540)
- Identify all of the substrings that form numbers that are divisible by 3.
- For example, applying the algorithm on the string 37540 should produce the following substrings (not necessarily in this order): 0; 3; 75; 54; 375; 540.

My solution to the problem:



```
def substrings_divisible_by_3(numberstring):  
    result = set()  
    for begin in range(0, len(numberstring)):  
        for end in range(begin+1, len(numberstring)+1):  
            subnumber = numberstring[begin:end]  
            if int(subnumber) % 3 == 0:  
                result.add(subnumber)  
    return result
```

Below are the tests I used to “drive” out and verify the above solution.



```
# 1-number string that is not divisible by 3 should return empty  
assert substrings_divisible_by_3("1") == set()  
# 1-number string that is divisible by 3 should return the number  
assert substrings_divisible_by_3("3") == {"3"}  
# 2-number string with 1 number divisible by 3 should return that number  
assert substrings_divisible_by_3("62") == {"6"}  
# 2-number string with both number divisible by 3 should return 3 numbers  
assert substrings_divisible_by_3("69") == {"6", "9", "69"}  
# test the example  
assert substrings_divisible_by_3("37540") == {"0", "3", "75", "54", "375", "540"}
```

## 2 Option 2: Word Search

- Make a list of five words, 3-6 letters in length.
- Create a string of approximately 30 letters containing some of the five words.
- Your algorithm should identify all of the substrings of the longer string that match any of the five words you generated, and the number of times each one appears
- For example: If you chose the words structure, such, system, blue, red, and your algorithm operates on the string jkdistructuredstrusyssystemoon, your algorithm should report that the string contains the words structure, red and system each one time, and the words such and blue zero times.



```
def count_words(text, words):  
    result = {}  
    for word in words:  
        result[word] = text.count(word)  
    return result
```

Below are the tests:



```
# should return empty dict when counting no word  
assert count_words("a string of apporximately 30 letters.", []) == {}  
# should count zero time when the string contains no word in the list  
assert count_words("a string of apporximately 30 letters.", ["precise", "1234"]) == {"precise":0, "1234":0}  
# should count 1 when the string contains 1 word in the list once  
assert count_words("a string of apporximately 30 letters.", ["of"]) == {"of":1}  
# acceptance  
assert count_words("jkdistructuredstrusyssystemoon", ["structure", "such", "system", "blue", "red"]) == {"structure":1, "such":0, "system":1, "blue":0, "red":1}
```


## 3 Option 3: Consensus Algorithm

Ten people need to decide which one flavour of ice they will order as a group. There are 3 types of ice cream from which to choose.

- Design an algorithm which can survey and re-survey each person, with the goal of reaching consensus on one kind of ice cream. The algorithm can present answers to each person in the group until a consensus is reached.
- This task is open-ended. You may make assumptions as needed. Explain these assumptions in your initial post.
- Determine whether there are situations in which your algorithm may never result in an answer, and account for this when writing your pseudo-code.

### 3.1 My Assumptions:

My assumptions are shown below in the form of test cases.



```

# if not survey answers are given to the algorithm, it should return all the options
assert consensus_icecream_flavor(None) == {"A", "B", "C"}


# if all the survey answers are the same, it should return consensus result
assert consensus_icecream_flavor(["A", "A", "A", "A", "A", "A", "A", "A", "A", "A"]) == {"A"}

# if different survey answers are given to the algorithm,
# it should return the options without the least favority one
assert consensus_icecream_flavor(["A", "A", "A", "A", "B", "B", "B", "B", "C", "C"]) == {"A", "B"}

# it should return all the options if there are more than one least favority flavor
assert consensus_icecream_flavor(["A", "A", "A", "A", "A", "A", "B", "B", "C", "C"]) == {"A", "B", "C"}

```

### 3.2 The solution derived from the above assumptions:



```

def consensus_icecream_flavor(answers):
    if answers is None:
        return {"A", "B", "C"}

    # get the votes of each flavor
    votes = {}
    for flavor in answers:
        if flavor not in votes:
            votes[flavor] = 0
        votes[flavor] += 1

    # reach consensus
    if len(votes) == 1:
        return {answers[0]}

    # get the least favorite flavor
    least_favorite = min(votes.values())

    # remove the least favorite flavor from the result
    result = {flavor for flavor in votes if votes[flavor] != least_favorite}

    # more than one least favorite
    if len(result) + 1 != len(votes):
        result = {flavor for flavor in votes}

    return result

```

Below is the consensus algorithm in action:



```
def icecream_survey():
    answers = None
    while True:
        options = consensus_icecream_flavor(answers)
        if len(options) == 1:
            return options[0] # reach consensus
        answers = []
        for index in range(10):
            answer = ask_to_choose_from_options(index, options)
            answers.append(answer)
```

The sub-routine `ask_to_choose_from_options(index, options)` will ask the user of `index` to choose one option from the `options`, and return that option.

### 3.3 Existing problems of this algorithm

This algorithm will never end with an answer if two flavors constantly get the same amount of votes. It's the situation described as the last piece in the assumptions above.

## References

Pérez, F. & Granger, B. E. (2007), 'IPython: a System for Interactive Scientific Computing', *Computing in Science & Engineering* 9(3), 21–29. URL: <http://ipython.org>.