Pre-final exam

COMP9021, Trimester 1, May 2019

- There are 8 templates, whose names are `exercise_1.py`, ..., `exercise_8.py`.
- There is a text file, whose name is `dictionary.txt`, that is needed for the 8th exercise.
- All templates make use of the `doctest` module and include a number of test cases. The purpose of the test cases is twofold:
  - describe by example what is expected;
  - let you easily test your programs as you develop them.
- When the test cases might not be enough to convey the expected behaviour, comments for a function are provided as a complement.
- Some templates might also include syntactic notes.
- Some programs require to insert more code than others.
- Tackle the questions in an order that is appropriate to you.
- You will be much better off if you do fewer questions but do them well, than if you attempt all questions but do nothing well. **So do what you can, but do it well.**
- For convenience, the test cases and other comments for each exercise are displayed in this pdf.

exercise_1.py

```
def rearrange(number):
    '''
    Returns an integer consisting of all nonzero digits in "number",
    from smallest to largest.

    You can assume that "number" is a valid strictly positive integer.

    >>> rearrange(1)
    1
    >>> rearrange(200)
    2
    >>> rearrange(395)
    359
    >>> rearrange(10029001)
    1129
    >>> rearrange(301302004)
    12334
    >>> rearrange(9409898038908908934890)
    3344888888999999
    '''
```

exercise_2.py

```
def rearrange(L, from_first = True):
    '''
    Returns a new list consisting of:
    * in case "from_first" is True:
         L's first member if it exists, then
         L's last member if it exists, then
         L's second member if it exists, then
         L's second last member if it exists, then
         L's third member if it exists...
    * in case "from_first" is False:
         L's last member if it exists, then
         L's first member if it exists, then
         L's second last member if it exists, then
         L's second member if it exists, then
         L's third last member if it exists...

    >>> L = []
    >>> rearrange(L), L
    ([], [])
    >>> L = [10]
    >>> rearrange(L, False), L
    ([10], [10])
    >>> L = [10, 20]
    >>> rearrange(L), L
    ([10, 20], [10, 20])
    >>> L = [10, 20, 30]
    >>> rearrange(L), L
    ([10, 30, 20], [10, 20, 30])
    >>> L = [10, 20, 30, 40]
    >>> rearrange(L, False), L
    ([40, 10, 30, 20], [10, 20, 30, 40])
    >>> L = [10, 20, 30, 40, 50]
    >>> rearrange(L, False), L
    ([50, 10, 40, 20, 30], [10, 20, 30, 40, 50])
    >>> L = [10, 20, 30, 40, 50, 60]
    >>> rearrange(L), L
    ([10, 60, 20, 50, 30, 40], [10, 20, 30, 40, 50, 60])
    >>> L = [10, 20, 30, 40, 50, 60, 70]
    >>> rearrange(L), L
    ([10, 70, 20, 60, 30, 50, 40], [10, 20, 30, 40, 50, 60, 70])
    '''
```

exercise_3.py

```
def single_factors(number):
    '''
    Returns the product of the prime divisors of "number"
    (using each prime divisor only once).

    You can assume that "number" is an integer at least equal to 2.

    >>> single_factors(2)
    2
    >>> single_factors(4096)                 # 4096 == 2**12
    2
    >>> single_factors(85)                   # 85 == 5 * 17
    85
    >>> single_factors(10440125)             # 10440125 == 5**3 * 17**4
    85
    >>> single_factors(154)                  # 154 == 2 * 7 * 11
    154
    >>> single_factors(52399401037149926144) # 52399401037149926144 == 2**8 * 7**2 * 11**15
    154
    '''
```

exercise_4.py

```
def words(text):
    '''
    You can assume that "text" contains nothing but words that consist of nothing but letters
    (no apostrophe, no hyphen, no quote symbol...), possibly followed by commas, full stops,
    colons, semicolons, question marks and exclamation marks (all with no space before, and
    with a space after if not the last character).
    Prints out lists of words with all uppercase letters changed to lowercase,
    with no duplicate, in lexicographic order, grouped together by number of letters,
    from smallest number of letters to largest number of letters.
    Observe that each list of words is preceded with 4 spaces.

    >>> words('Twelve will, believe me, be quite enough for your purpose.')
    Words of length 2:
        ['be', 'me']
    Words of length 3:
        ['for']
    Words of length 4:
        ['will', 'your']
    Words of length 5:
        ['quite']
    Words of length 6:
        ['enough', 'twelve']
    Words of length 7:
        ['believe', 'purpose']
    >>> words('What was that? What does the Bishop mean?')
    Words of length 3:
        ['the', 'was']
    Words of length 4:
        ['does', 'mean', 'that', 'what']
    Words of length 6:
        ['bishop']
    >>> words('You must not fall into the common error of mistaking these simpletons '\
              'for liars and hypocrites. They believe honestly and sincerely that their '\
              'diabolical inspiration is divine. Therefore you must be on your guard against '\
              'your natural compassion. You are all, I hope, merciful men: how else could you '\
              'have devoted your lives to the service of our gentle Savior? You are going '\
              'to see before you a young girl, pious and chaste; for I must tell you, gentlemen, '\
              'that the things said of her by our English friends are supported by no evidence, '\
              'whilst there is abundant testimony that her excesses have been excesses of religion '\
              'and charity and not of worldliness and wantonness.')
    Words of length 1:
        ['a', 'i']
    Words of length 2:
        ['be', 'by', 'is', 'no', 'of', 'on', 'to']
    Words of length 3:
        ['all', 'and', 'are', 'for', 'her', 'how', 'men', 'not', 'our', 'see', 'the', 'you']
    Words of length 4:
        ['been', 'else', 'fall', 'girl', 'have', 'hope', 'into', 'must', 'said', 'tell', 'that', 'they', 'your']
    Words of length 5:
        ['could', 'error', 'going', 'guard', 'liars', 'lives', 'pious', 'their', 'there', 'these', 'young']
    Words of length 6:
        ['before', 'chaste', 'common', 'divine', 'gentle', 'savior', 'things', 'whilst']
    Words of length 7:
        ['against', 'believe', 'charity', 'devoted', 'english', 'friends', 'natural', 'service']
    Words of length 8:
        ['abundant', 'evidence', 'excesses', 'honestly', 'merciful', 'religion']
```

```
Words of length 9:
    ['gentlemen', 'mistaking', 'sincerely', 'supported', 'testimony', 'therefore']
Words of length 10:
    ['compassion', 'diabolical', 'hypocrites', 'simpletons', 'wantonness']
Words of length 11:
    ['inspiration', 'worldliness']
'''
```

exercise_5.py

```
# You might find the ord() function useful.

def longest_leftmost_sequence_of_consecutive_letters(word):
    '''
    You can assume that "word" is a string of
    nothing but lowercase letters.

    >>> longest_leftmost_sequence_of_consecutive_letters('')
    ''
    >>> longest_leftmost_sequence_of_consecutive_letters('a')
    'a'
    >>> longest_leftmost_sequence_of_consecutive_letters('zuba')
    'z'
    >>> longest_leftmost_sequence_of_consecutive_letters('ab')
    'ab'
    >>> longest_leftmost_sequence_of_consecutive_letters('bcab')
    'bc'
    >>> longest_leftmost_sequence_of_consecutive_letters('aabbccddee')
    'ab'
    >>> longest_leftmost_sequence_of_consecutive_letters('aefbxyzcrsdt')
    'xyz'
    >>> longest_leftmost_sequence_of_consecutive_letters('efghuvwijlrstuvabcde')
    'rstuv'
    '''
```

exercise_6.py

```
# ord(c) returns the encoding of character c.
# chr(e) returns the character encoded by e.

def rectangle(width, height):
    '''
    Displays a rectangle by outputting lowercase letters, starting with a,
    in a "snakelike" manner, from left to right, then from right to left,
    then from left to right, then from right to left, wrapping around when z is reached.

    >>> rectangle(1, 1)
    a
    >>> rectangle(2, 3)
    ab
    dc
    ef
    >>> rectangle(3, 2)
    abc
    fed
    >>> rectangle(17, 4)
    abcdefghijklmnopq
    hgfedcbazyxwvutsr
    ijklmnopqrstuvwxy
    ponmlkjihgfedcbaz
    '''
```

exercise_7.py

```
def subnumbers_whose_digits_add_up_to(number, sum_of_digits):
    '''
    You can assume that "number" consists of digits not equal to 0
    and that "sum_of_digits" is an integer.

    A solution is obtained by possibly deleting some digits in "number"
    (keeping the order of the remaining digits) so that the sum of
    of the remaining digits is equal to "sum_of_digits".

    The solutions are listed from smallest to largest, with no duplicate.

    >>> subnumbers_whose_digits_add_up_to(13, 2)
    []
    >>> subnumbers_whose_digits_add_up_to(222, 2)
    [2]
    >>> subnumbers_whose_digits_add_up_to(123, 6)
    [123]
    >>> subnumbers_whose_digits_add_up_to(222, 4)
    [22]
    >>> subnumbers_whose_digits_add_up_to(1234, 5)
    [14, 23]
    >>> subnumbers_whose_digits_add_up_to(12341234, 4)
    [4, 13, 22, 31, 112, 121]
    >>> subnumbers_whose_digits_add_up_to(121212, 5)
    [122, 212, 221, 1112, 1121, 1211]
    >>> subnumbers_whose_digits_add_up_to(123454321, 10)
    [145, 154, 235, 244, 253, 343, 352, 442, 451, 532, 541, 1234, 1243, \
1252, 1342, 1351, 1432, 1441, 1531, 2332, 2341, 2431, 2521, 3421, \
4321, 12331, 12421, 13321]
    '''
```

exercise_8.py

```
def number_of_words_in_dictionary(word_1, word_2):
    '''
    "dictionary.txt" is stored in the working directory.

    >>> number_of_words_in_dictionary('company', 'company')
    Could not find company in dictionary.
    >>> number_of_words_in_dictionary('company', 'comparison')
    Could not find at least one of company and comparison in dictionary.
    >>> number_of_words_in_dictionary('COMPANY', 'comparison')
    Could not find at least one of COMPANY and comparison in dictionary.
    >>> number_of_words_in_dictionary('company', 'COMPARISON')
    Could not find at least one of company and COMPARISON in dictionary.
    >>> number_of_words_in_dictionary('COMPANY', 'COMPANY')
    COMPANY is in dictionary.
    >>> number_of_words_in_dictionary('COMPARISON', 'COMPARISON')
    COMPARISON is in dictionary.
    >>> number_of_words_in_dictionary('COMPANY', 'COMPARISON')
    Found 14 words between COMPANY and COMPARISON in dictionary.
    >>> number_of_words_in_dictionary('COMPARISON', 'COMPANY')
    Found 14 words between COMPARISON and COMPANY in dictionary.
    >>> number_of_words_in_dictionary('CONSCIOUS', 'CONSCIOUSLY')
    Found 2 words between CONSCIOUS and CONSCIOUSLY in dictionary.
    >>> number_of_words_in_dictionary('CONSCIOUS', 'CONSCIENTIOUS')
    Found 3 words between CONSCIOUS and CONSCIENTIOUS in dictionary.
    '''
```