Sample Exam

COMP9021, Term 3, 2019

- There are 8 templates, whose names are `exercise_1.py`, ..., `exercise_8.py`.
- 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 are not enough to make it clear what is expected, comments for a function are provided as a complement.
- Some templates 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 what you do well.
- For convenience, the test cases and other comments for each exercise are displayed in this pdf.

1 exercise_1.py

```
>>> f(0, 0, 10)
Here is L: []
The decomposition of L into increasing sequences,
    with consecutive duplicates removed, is:
    []
>>> f(0, 1, 10)
Here is L: [6]
The decomposition of L into increasing sequences,
    with consecutive duplicates removed, is:
    [[6]]
>>> f(0, 2, 10)
Here is L: [6, 6]
The decomposition of L into increasing sequences,
    with consecutive duplicates removed, is:
    [[6]]
>>> f(0, 3, 10)
Here is L: [6, 6, 0]
The decomposition of L into increasing sequences,
    with consecutive duplicates removed, is:
    [[6], [0]]
>>> f(0, 4, 10)
Here is L: [6, 6, 0, 4]
The decomposition of L into increasing sequences,
    with consecutive duplicates removed, is:
    [[6], [0, 4]]
>>> f(0, 5, 10)
Here is L: [6, 6, 0, 4, 8]
The decomposition of L into increasing sequences,
    with consecutive duplicates removed, is:
    [[6], [0, 4, 8]]
>>> f(0, 6, 10)
Here is L: [6, 6, 0, 4, 8, 7]
The decomposition of L into increasing sequences,
    with consecutive duplicates removed, is:
    [[6], [0, 4, 8], [7]]
>>> f(0, 7, 10)
Here is L: [6, 6, 0, 4, 8, 7, 6]
The decomposition of L into increasing sequences,
    with consecutive duplicates removed, is:
    [[6], [0, 4, 8], [7], [6]]
```

```
>>> f(3, 10, 6)
Here is L: [1, 4, 4, 1, 2, 4, 3, 5, 4, 0]
The decomposition of L into increasing sequences,
    with consecutive duplicates removed, is:
    [[1, 4], [1, 2, 4], [3, 5], [4], [0]]
>>> f(3, 15, 8)
Here is L: [3, 8, 2, 5, 7, 1, 0, 7, 4, 8, 3, 3, 7, 8, 8]
The decomposition of L into increasing sequences,
    with consecutive duplicates removed, is:
    [[3, 8], [2, 5, 7], [1], [0, 7], [4, 8], [3, 7, 8]]
```

2 exercise_2.py

```
You might find the function bin() useful.
Will be tested with n a strictly positive integer.
```

```
    >>> f(1)
    1 in binary reads as: 1.
    Only one bit is set to 1 in the binary representation of 1.
    >>> f(2)
    2 in binary reads as: 10.
    Only one bit is set to 1 in the binary representation of 2.
    >>> f(3)
    3 in binary reads as: 11.
    2 bits are set to 1 in the binary representation of 3.
    >>> f(7)
    7 in binary reads as: 111.
    3 bits are set to 1 in the binary representation of 7.
    >>> f(2314)
    2314 in binary reads as: 100100001010.
    4 bits are set to 1 in the binary representation of 2314.
    >>> f(9871)
    9871 in binary reads as: 10011010001111.
    8 bits are set to 1 in the binary representation of 9871.
```

3 exercise_3.py

Will be tested with n at least equal to 2, and "not too large".

```
>>> f(2)
The decomposition of 2 into prime factors reads:
   2 = 2
>>> f(3)
The decomposition of 3 into prime factors reads:
   3 = 3
>>> f(4)
The decomposition of 4 into prime factors reads:
   4 = 2^2
>>> f(5)
The decomposition of 5 into prime factors reads:
   5 = 5
>>> f(6)
The decomposition of 6 into prime factors reads:
   6 = 2 x 3
>>> f(8)
The decomposition of 8 into prime factors reads:
   8 = 2^3
>>> f(10)
The decomposition of 10 into prime factors reads:
   10 = 2 x 5
>>> f(15)
The decomposition of 15 into prime factors reads:
   15 = 3 x 5
>>> f(100)
The decomposition of 100 into prime factors reads:
   100 = 2^2 x 5^2
>>> f(5432)
The decomposition of 5432 into prime factors reads:
   5432 = 2^3 x 7 x 97
>>> f(45103)
The decomposition of 45103 into prime factors reads:
   45103 = 23 x 37 x 53
>>> f(45100)
The decomposition of 45100 into prime factors reads:
   45100 = 2^2 x 5^2 x 11 x 41
```

4 exercise_4.py

Will be tested with a at equal equal to 2 and b at most equal to 10_000_000.

```
>>> f(2, 2)
There is a unique prime number beween 2 and 2.
>>> f(2, 3)
There are 2 prime numbers between 2 and 3.
>>> f(2, 5)
There are 3 prime numbers between 2 and 5.
>>> f(4, 4)
There is no prime number beween 4 and 4.
>>> f(14, 16)
There is no prime number beween 14 and 16.
>>> f(3, 20)
There are 7 prime numbers between 3 and 20.
>>> f(100, 800)
There are 114 prime numbers between 100 and 800.
>>> f(123, 456789)
There are 38194 prime numbers between 123 and 456789.
```

5 exercise_5.py

```
Will be tested with year between 1913 and 2013.
You might find the reader() function of the csv module useful,
but you can also use the split() method of the str class.
```

```
 >>> f(1914)
In 1914, maximum inflation was: 2.0
It was achieved in the following months: Aug
>>> f(1922)
In 1922, maximum inflation was: 0.6
It was achieved in the following months: Jul, Oct, Nov, Dec
>>> f(1995)
In 1995, maximum inflation was: 0.4
It was achieved in the following months: Jan, Feb
>>> f(2013)
In 2013, maximum inflation was: 0.82
It was achieved in the following months: Feb
```

6 exercise_6.py

You might find the zip() function useful, though you can also do without it.

```
>>> f(0, 2, 2)
Here is the square:
1 1
0 1
It is not a good square because it contains duplicates, namely: 1
>>> f(0, 3, 5)
Here is the square:
3 3 0
2 4 3
3 2 3
It is not a good square because it contains duplicates, namely: 2 3
>>> f(0, 6, 50)
Here is the square:
24 48 26  2 16 32
31 25 19 30 22 37
13 32  8 18  8 48
 6 39 16 34 45 38
 9 19  6 46  4 43
21 30 35  6 22 27
It is not a good square because it contains duplicates, namely:...
                                                        ...6 8 16 19 22 30 32 48
>>> f(0, 2, 50)
Here is the square:
24 48
26  2
It is a good square.
Ordering the elements from left to right column, from top to bottom, yields:
 2 26
24 48
>>> f(0, 3, 100)
Here is the square:
 49  97  53
  5  33  65
 62  51  38
It is a good square.
Ordering the elements from left to right column, from top to bottom, yields:
  5  49  62
 33  51  65
 38  53  97
```

```
>>> f(0, 6, 5000)
Here is the square:
3155 3445  331 2121 4188 3980
3317 2484 3904 2933 4779 1789
4134 1140 2308 1144  776 2052
4362 4930 1203 2540  809  604
2704 3867 4585  824 2898 3556
2590 1675 4526 3907 3626 4270
It is a good square.
Ordering the elements from left to right column, from top to bottom, yields:
 331 1144 2308 2933 3867 4270
 604 1203 2484 3155 3904 4362
 776 1675 2540 3317 3907 4526
 809 1789 2590 3445 3980 4585
 824 2052 2704 3556 4134 4779
1140 2121 2898 3626 4188 4930
```

7 exercise_7.py

Will be tested with height a strictly positive integer.

```
    >>> f(1)
    0
    >>> f(2)
     0
    123
    >>> f(3)
      0
     123
    45678
    >>> f(4)
       0
      123
     45678
    9012345
    >>> f(5)
         0
        123
       45678
      9012345
    678901234
    >>> f(6)
          0
         123
        45678
       9012345
      678901234
    56789012345
```

```
>>> f(20)
                    0
                   123
                  45678
                 9012345
                678901234
               56789012345
              6789012345678
             901234567890123
            45678901234567890
           1234567890123456789
          012345678901234567890
         123456789012345678901123
        45678901234567890123456789
       9012345678901234567890123456
      678901234567890123456789012345
     56789012345678901234567890123456
    6789012345678901234567890123456789
  901234567890123456789012345678901234567
 45678901234567890123456789012345678901234
1234567890123456789012345678901234567890123
```

8 exercise_8.py

Will be tested with letters a string of DISTINCT UPPERCASE letters only.

```
>>> f('ABCDEFGH')
There is no solution
>>> f('GRIHWSNYP')
The pairs of words using all (distinct) letters in "GRIHWSNYP" are:
('SPRING', 'WHY')
>>> f('ONESIX')
The pairs of words using all (distinct) letters in "ONESIX" are:
('ION', 'SEX')
('ONE', 'SIX')
```

```
>>> f('UTAROFSMN')
The pairs of words using all (distinct) letters in "UTAROFSMN" are:
('AFT', 'MOURNS')
('ANT', 'FORUMS')
('ANTS', 'FORUM')
('ARM', 'FOUNTS')
('ARMS', 'FOUNT')
('AUNT', 'FORMS')
('AUNTS', 'FORM')
('AUNTS', 'FROM')
('FAN', 'TUMORS')
('FANS', 'TUMOR')
('FAR', 'MOUNTS')
('FARM', 'SNOUT')
('FARMS', 'UNTO')
('FAST', 'MOURN')
('FAT', 'MOURNS')
('FATS', 'MOURN')
('FAUN', 'STORM')
('FAUN', 'STROM')
('FAUST', 'MORN')
('FAUST', 'NORM')
('FOAM', 'TURNS')
('FOAMS', 'RUNT')
('FOAMS', 'TURN')
('FORMAT', 'SUN')
('FORUM', 'STAN')
('FORUMS', 'NAT')
('FORUMS', 'TAN')
('FOUNT', 'MARS')
('FOUNT', 'RAMS')
('FOUNTS', 'RAM')
('FUR', 'MATSON')
('MASON', 'TURF')
('MOANS', 'TURF')
```