Mid-term exam

COMP9021, Session 2, September 2017

- 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 sublists is: []
>>> f(0, 1, 10)
Here is L: [6]
The decomposition of L into increasing sublists is: [[6]]
>>> f(0, 2, 10)
Here is L: [6, 6]
The decomposition of L into increasing sublists is: [[6], [6]]
>>> f(1, 2, 10)
Here is L: [2, 9]
The decomposition of L into increasing sublists is: [[2, 9]]
>>> f(0, 3, 10)
Here is L: [6, 6, 0]
The decomposition of L into increasing sublists is: [[6], [6], [0]]
>>> f(1, 4, 10)
Here is L: [2, 9, 1, 4]
The decomposition of L into increasing sublists is: [[2, 9], [1, 4]]
>>> f(20, 5, 10)
Here is L: [10, 2, 4, 10, 10]
The decomposition of L into increasing sublists is: [[10], [2, 4, 10], [10]]
>>> f(1, 10, 20)
Here is L: [4, 18, 2, 8, 3, 15, 14, 15, 20, 12]
The decomposition of L into increasing sublists is: [[4, 18], [2, 8], [3, 15],
                                                          [14, 15, 20], [12]]
```

2 exercise_2.py

```
>>> f(10, 0, 1)
Here is L: []
>>> f(0, 1, 1)
Here is L: [6]
List members of length 1 with no successive occurrences of the same digit:
    6
>>> f(8, 6, 4)
Here is L: [5, 129, 0, 0, 8, 6]
List members of length 1 with no successive occurrences of the same digit:
    5 0 0 8 6
List members of length 3 with no successive occurrences of the same digit:
    129
>>> f(20, 8, 5)
Here is L: [89948, 4, 83325, 0, 2775, 0, 76, 0]
List members of length 1 with no successive occurrences of the same digit:
    4 0 0 0
List members of length 2 with no successive occurrences of the same digit:
    76
>>> f(30, 10, 7)
Here is L: [492, 263, 0, 672743, 10, 127618, 26, 2, 872293, 70150]
List members of length 1 with no successive occurrences of the same digit:
    0 2
List members of length 2 with no successive occurrences of the same digit:
    10 26
List members of length 3 with no successive occurrences of the same digit:
    492 263
List members of length 5 with no successive occurrences of the same digit:
    70150
List members of length 6 with no successive occurrences of the same digit:
    672743 127618
>>> f(30, 12, 5)
Here is L: [4738, 492, 3440, 6, 385, 17572, 0, 0, 0, 9, 6582, 45665]
List members of length 1 with no successive occurrences of the same digit:
    6 0 0 0 9
List members of length 3 with no successive occurrences of the same digit:
    492 385
List members of length 4 with no successive occurrences of the same digit:
    4738 6582
List members of length 5 with no successive occurrences of the same digit:
    17572
```

3 exercise_3.py

```
>>> f(0)
0 factorial is 1
It has 1 digit, the trailing 0’s excepted
>>> f(4)
4 factorial is 24
It has 2 digits, the trailing 0’s excepted
>>> f(6)
6 factorial is 720
It has 2 digits, the trailing 0’s excepted
>>> f(10)
10 factorial is 3628800
It has 5 digits, the trailing 0’s excepted
>>> f(20)
20 factorial is 2432902008176640000
It has 15 digits, the trailing 0’s excepted
>>> f(30)
30 factorial is 265252859812191058636308480000000
It has 26 digits, the trailing 0’s excepted
>>> f(40)
40 factorial is 815915283247897734345611269596115894272000000000
It has 39 digits, the trailing 0’s excepted
```

4 exercise_4.py

```
A pair of natural numbers (p, q) is a Betrothed pair if:
- the sum of the proper divisors of p is one more than q;
- the sum of the proper divisors of q is one more than p.
For instance, (48, 75) is a Betrothed pair since
- the proper divisors of 48 are 1, 2, 3, 4, 6, 8, 12, 16 and 24
- the proper divisors of 75 are 1, 3, 5, 15 and 25
- 1 + 2 + 3 + 4 + 6 + 8 + 12 + 16 + 24 = 76
- 1 + 3 + 5 + 15 + 25 = 49

Won't be tested for n greater than 10_000

>>> f(0)
The least number >= 0 that is the first member of a Betrothed pair is 48
The Betrothed pair itself is (48, 75)
>>> f(50)
The least number >= 50 that is the first member of a Betrothed pair is 75
The Betrothed pair itself is (75, 48)
>>> f(100)
The least number >= 100 that is the first member of a Betrothed pair is 140
The Betrothed pair itself is (140, 195)
```

5 exercise_5.py

```
The prime numbers between 2 and 12 (both included) are: 2, 3, 5, 7, 11
The gaps between successive primes are: 0, 1, 1, 3.
Hence the maximum gap is 3.

Won't be tested for second argument greater than 10_000_000

>>> f(3, 3)
The maximum gap between successive prime numbers in that interval is 0
>>> f(3, 4)
The maximum gap between successive prime numbers in that interval is 0
>>> f(3, 5)
The maximum gap between successive prime numbers in that interval is 1
>>> f(2, 12)
The maximum gap between successive prime numbers in that interval is 3
>>> f(5, 23)
The maximum gap between successive prime numbers in that interval is 3
>>> f(20, 106)
The maximum gap between successive prime numbers in that interval is 7
>>> f(31, 291)
The maximum gap between successive prime numbers in that interval is 13
```

6 exercise_6.py

```
Finds all numbers i and j with a <= i <= j <= b such that:
- i + j is even;
- when read from left to right, the digits in i are strictly increasing
- when read from left to right, the digits in j are strictly decreasing
- when read from left to right, the digits in the average of i and j are
  either strictly increasing or strictly decreasing

Outputs the solutions from smallest i to largest i,
and for a given i from smallest j to largest j.

>>> f(10, 20)
12 and 20 with 16 as average
14 and 20 with 17 as average
16 and 20 with 18 as average
18 and 20 with 19 as average
>>> f(30, 50)
34 and 40 with 37 as average
34 and 42 with 38 as average
34 and 50 with 42 as average
35 and 41 with 38 as average
35 and 43 with 39 as average
36 and 40 with 38 as average
36 and 42 with 39 as average
36 and 50 with 43 as average
37 and 41 with 39 as average
37 and 43 with 40 as average
38 and 40 with 39 as average
38 and 42 with 40 as average
39 and 41 with 40 as average
39 and 43 with 41 as average
46 and 50 with 48 as average
48 and 50 with 49 as average
>>> f(400, 700)
456 and 630 with 543 as average
457 and 521 with 489 as average
458 and 520 with 489 as average
459 and 621 with 540 as average
468 and 510 with 489 as average
478 and 542 with 510 as average
479 and 541 with 510 as average
489 and 531 with 510 as average
567 and 653 with 610 as average
568 and 610 with 589 as average
568 and 652 with 610 as average
569 and 651 with 610 as average
578 and 642 with 610 as average
579 and 641 with 610 as average
589 and 631 with 610 as average
589 and 651 with 620 as average
589 and 653 with 621 as average
```

7 exercise_7.py

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

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.

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

8 exercise_8.py

```
A heterosquare of order n is an arrangement of the integers 1 to n**2 in a square,
such that the rows, columns, and diagonals all sum to DIFFERENT values.
In contrast, magic squares have all these sums equal.


>>> is_heterosquare([[1, 2, 3],\
                     [8, 9, 4],\
                     [7, 6, 5]])
True
>>> is_heterosquare([[1, 2, 3],\
                     [9, 8, 4],\
                     [7, 6, 5]])
False
>>> is_heterosquare([[2, 1, 3, 4],\
                     [5, 6, 7, 8],\
                     [9, 10, 11, 12],\
                     [13, 14, 15, 16]])
True
>>> is_heterosquare([[1, 2, 3, 4],\
                     [5, 6, 7, 8],\
                     [9, 10, 11, 12],\
                     [13, 14, 15, 16]])
False
```