Assignment 1

COMP9021, Session 2, 2018

1. General matters

1.1. **Aims.** The purpose of the assignment is to:

- let you design solutions to simple problems;
- let you implement these solutions in the form of short Python programs;
- practice the use of arithmetic computations, tests, repetitions, lists, tuples, dictionaries, deques, reading from files.

1.2. **Submission.** Your programs will be stored in a number of files, with one file per exercise, of the appropriate name. After you have developed and tested your programs, upload your files using Ed. Assignments can be submitted more than once: the last version is marked. Your assignment is due by September 2, 11:59pm.

1.3. **Assessment.** Each of the 4 exercises is worth 2.5 marks. For all exercises, the automarking script will let each of your programs run for 30 seconds. Still you should not take advantage of this and strive for a solution that gives an immediate output for "reasonable" inputs.

Late assignments will be penalised: the mark for a late submission will be the minimum of the awarded mark and 10 minus the number of full and partial days that have elapsed from the due date.

The outputs of your programs should be **exactly** as indicated.

1.4. **Reminder on plagiarism policy.** You are permitted, indeed encouraged, to discuss ways to solve the assignment with other people. Such discussions must be in terms of algorithms, not code. But you must implement the solution on your own. Submissions are routinely scanned for similarities that occur when students copy and modify other people's work, or work very closely together on a single implementation. Severe penalties apply.

2. Rain

Write a program, stored in a file named `rain.py`, that performs the following task.

- The program prompts the user to input a file name. If there is no file with that name in the working directory, then the program outputs an (arbitrary) error message and exits.
- The program prompts the user to input a nonnegative integer. If the input is not a nonnegative integer, then the program outputs an (arbitrary) error message and exits.
- The contents of the file consists of some number of lines, each line being a sequence of the same number of nonnegative integers separated by at least one space, with possibly spaces before and after the first and last number, respectively. These numbers represent the height in centimetres of a block with a square base of 10 square centimetres. For instance, the contents of the file `land.txt` can be represented as

2	2	2	2	2
1	3	4	3	2
2	3	5	3	2
2	4	1	1	2

  and would represent a land covering an area of 50 centimetres by 40 centimetres, with a block of height 2 centimetres in each corner, etc. The number input by the user represents a quantity of rain in decilitres, to be poured down on the land.
- The program outputs the height in centimetres, with a precision of 2 digits after the decimal point, that is reached by the water.

Here is a possible interaction:

```
$ cat land.txt
2  2  2  2  2
1  3  4  3  2
2  3  5  3  2
2  4  1  1  2
$ python3 rain.py
Which data file do you want to use? land.txt
How many decilitres of water do you want to pour down? 1
The water rises to 1.33 centimetres.
$ $ python3 rain.py
Which data file do you want to use? $ land.txt
How many decilitres of water do you want to pour down? 33
The water rises to 4.00 centimetres.
$ $ python3 rain.py
Which data file do you want to use? $ land.txt
How many decilitres of water do you want to pour down? 50
The water rises to 4.89 centimetres.
```

You can assume that the contents of any test file is as expected, you do not have to check that it is as expected.

3. Superpowers

Write a program, stored in a file named `superpower.py`, that does the following.

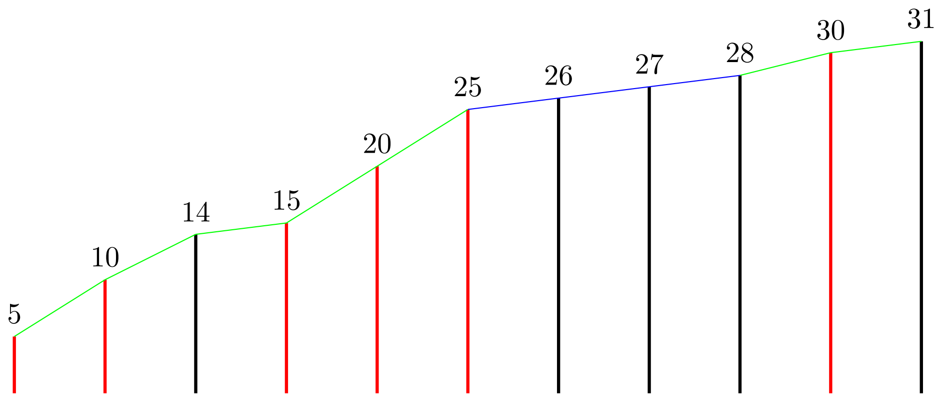
- The program prompts the user for an arbitrary number of (positive or negative) integers, all provided on one line and separated by at least one space, with possibly extra spaces anywhere. The user can input no integer at all. If the input is incorrect, then the program outputs an error message and exits. These integers are meant to represent the superpower of a bunch of heroes (positive power is good, all the better that it is larger; negative power is bad, all the worse that it is smaller).
- Then the program prompts the user for a nonnegative integer (possibly equal to 0), `nb_of_switches`, at most equal to the number of integers previously input (the number of heroes). If the input is incorrect, then the program outputs another error message and exits. This number is meant to represent the number of switches (multiplication by -1) to the power of some heroes; for instance, a hero with positive power 7 can see its power switched to -7, and a hero with negative power -5 can see its power switched to 5.
- First, the program determines the maximum overall power that can be achieved by switching the power of `nb_of_switches` many heroes, a given hero seeing its power being switched an arbitrary number of times (not switched at all, or switched once, or switched more than once). For instance, with the sequence (4, -3, 1, 2, -7) and `nb_of_switches` set to 3, the maximum overall power that can be achieved is 15; this can be done by switching -3, 1 and -7 once: $4 + 3 - 1 + 2 + 7 = 15$. For another example, with the sequence (-7, 1, 2, 3, 4) and `nb_of_switches` set to 5, the maximum overall power that can be achieved is 17; this can be done by switching -7 once, 1 twice, and 3 twice: $7 + 1 + 2 + 3 + 4 = 17$.
- Second, the program determines the maximum overall power that can be achieved by switching the power of `nb_of_switches` many heroes, a given hero seeing its power being switched at most once (not switched at all, or switched once). For instance, with the sequence (4, -3, 1, 2, -7) and `nb_of_switches` set to 3, the maximum overall power that can be achieved is 15, as previously described. For another example, with the sequence (-7, 1, 2, 3, 4) and `nb_of_switches` set to 5, the maximum overall power that can be achieved is -3, as each power has to be switched: $7 - 1 - 2 - 3 - 4 = -3$.
- Third, the program determines the maximum overall power that can be achieved by switching the power of `nb_of_switches` many consecutive heroes (note that first and last heroes are not consecutive, the heroes are aligned on a line). For instance, with the sequence (4, -3, 1, 2, -7) and `nb_of_switches` set to 3, the maximum overall power that can be achieved is 5; this can be done by switching the power of the last 3 heroes: $4 - 3 - 1 - 2 + 7 = 5$. For another example, with the sequence (-7, 1, 2, 3, 4) and `nb_of_switches` set to 5, the maximum overall power that can be achieved is -3, as previously described.
- Fourth, the program determines the maximum overall power that can be achieved by switching the power of arbitrarily many (possibly none) consecutive heroes. For instance, with the sequence (4, -3, 1, 2, -7), the maximum overall power that can be achieved is 17; this can be done by switching the power of the last 4 heroes: $4 + 3 - 1 - 2 + 7 = 11$. For another example, with the sequence (-7, 1, 2, 3, 4), the maximum overall power that can be achieved is 17; this can be done by switching the power of the first hero: $7 + 1 + 2 + 3 + 4 = 17$.

Here is a possible interaction:

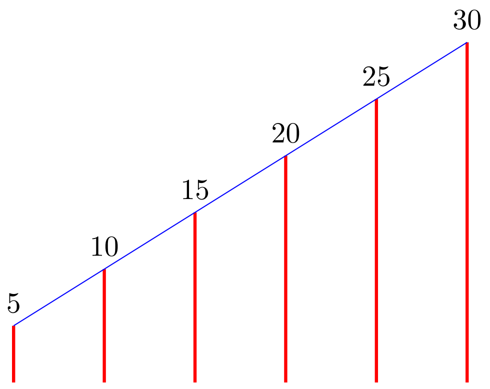
```
$ python3 superpower.py
Please input the heroes' powers: 3 5 8.5
Sorry, these are not valid power values.
$ python3 superpower.py
Please input the heroes' powers: 3 5 8
Please input the number of power flips: 4
Sorry, this is not a valid number of power flips.
$ python3 superpower.py
Please input the heroes' powers: 4 -3 1 2 -7
Please input the number of power flips: 3
Possibly flipping the power of the same hero many times, the greatest achievable power is 15.
Flipping the power of the same hero at most once, the greatest achievable power is 15.
Flipping the power of nb_of_flips many consecutive heroes, the greatest achievable power is 5.
Flipping the power of arbitrarily many consecutive heroes, the greatest achievable power is 11.
$ python3 superpower.py
Please input the heroes' powers: -7 1 2 3 4
Please input the number of power flips: 5
Possibly flipping the power of the same hero many times, the greatest achievable power is 17.
Flipping the power of the same hero at most once, the greatest achievable power is -3.
Flipping the power of nb_of_flips many consecutive heroes, the greatest achievable power is -3.
Flipping the power of arbitrarily many consecutive heroes, the greatest achievable power is 17.
$ python3 superpower.py
Please input the heroes' powers: 1 2 3 4 5
Please input the number of power flips: 2
Possibly flipping the power of the same hero many times, the greatest achievable power is 15.
Flipping the power of the same hero at most once, the greatest achievable power is 9.
Flipping the power of nb_of_flips many consecutive heroes, the greatest achievable power is 9.
Flipping the power of arbitrarily many consecutive heroes, the greatest achievable power is 15.
```

4. Cable car

Consider the following picture of the pillars and cable that provide the structure for a cable car. Successive pillars are equidistant. The height of each pillar is indicated above it. Continuous pillars make up for a good ride if the slope does not change over the corresponding section. Here the longest good ride is between the pillar of height 25 and the pillar of height 28; it is depicted in blue. It has a length of 3 (units, whatever it is).



If one gets rid of the black pillars and keeps only the remaining (red) ones, then one can use use the latter and create a new structure (with successive pillars being equidistant) which makes up for a perfect ride, that is, good from first to last pillar:



It is the longest perfect ride one can obtain this way; so 5, the number of black pillars, is the minimal number of pillars to remove to build a perfect ride from the rest.

Write a program, stored in a file named `cable_car.py`, that performs the following task.

- The program prompts the user to input a file name. If there is no file with that name in the working directory, then the program outputs an error message and exits.
- The contents of the file should consist of a strictly increasing sequence of at least two strictly positive integers. Consecutive numbers are separated by at least one space, and there can be extra spaces, including empty lines, anywhere. If that is not the case then the program outputs another error message and exits.
- These numbers are meant to represent the heights of pillars to build a cable car structure. The program then outputs:
  - whether or not this structure makes up for a perfect ride;
  - the length of the longest good ride;
  - how many pillars to remove to build a perfect ride from the remaining ones.

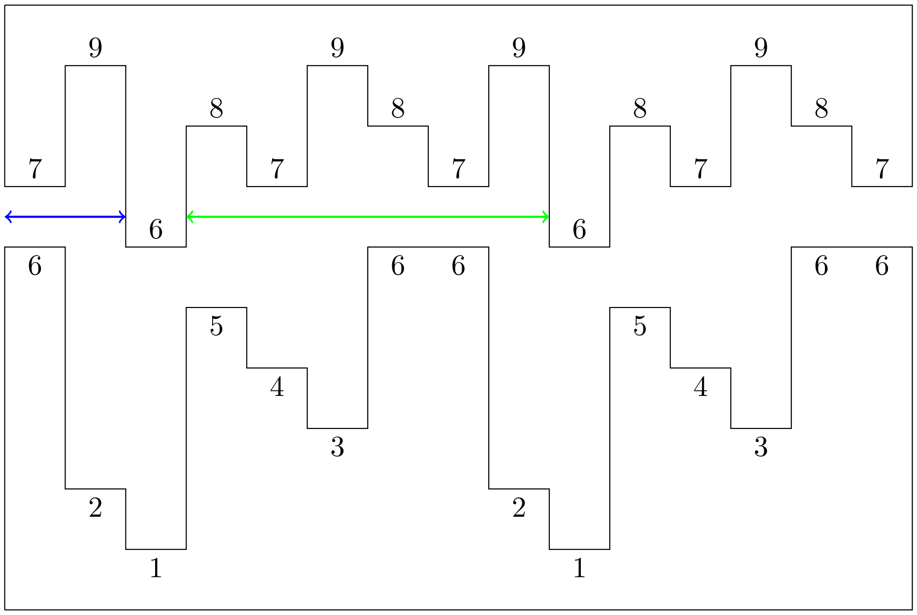
Here is a possible interaction:

```
$ python3 cable_car.py
Please enter the name of the file you want to get data from: non_existent_file.txt
Sorry, there is no such file.
$ cat cable_car_wrong_1.txt
 6
$ python3 cable_car.py
Please enter the name of the file you want to get data from: cable_car_wrong_1.txt
Sorry, input file does not store valid data.
$ cat cable_car_wrong_2.txt
 6 8 10 A 12
$ python3 cable_car.py
Please enter the name of the file you want to get data from: cable_car_wrong_2.txt
Sorry, input file does not store valid data.
$ cat cable_car_wrong_3.txt
0 2 4 6 8
$ python3 cable_car.py
Please enter the name of the file you want to get data from: cable_car_wrong_3.txt
Sorry, input file does not store valid data.
$ cat cable_car_wrong_4.txt
0 2 4 6 6 8
$ python3 cable_car.py
Please enter the name of the file you want to get data from: cable_car_wrong_4.txt
Sorry, input file does not store valid data.
$ cat cable_car_wrong_5.txt
0 2 4 6 5 8
$ python3 cable_car.py
Please enter the name of the file you want to get data from: cable_car_wrong_5.txt
Sorry, input file does not store valid data.
```

```
$ cat cable_car_1.txt
5  8
  11    14
$ python3 cable_car.py
Please enter the name of the file you want to get data from: cable_car_1.txt
The ride is perfect!
The longest good ride has a length of: 3
The minimal number of pillars to remove to build a perfect ride from the rest is: 0
$ cat cable_car_2.txt
  5
10 14 15


     20  25  26 27 28    30
  31
$ python3 cable_car.py
Please enter the name of the file you want to get data from: cable_car_2.txt
The ride could be better...
The longest good ride has a length of: 3
The minimal number of pillars to remove to build a perfect ride from the rest is: 5
$ cat cable_car_3.txt

10 13 20 30 40 42

     44 46 48 50 60 70 80 82 85 87
          90 100 101 110 113 117   121

$ python3 cable_car.py
Please enter the name of the file you want to get data from: cable_car_3.txt
The ride could be better...
The longest good ride has a length of: 5
The minimal number of pillars to remove to build a perfect ride from the rest is: 12
```

5. Tunnel

Consider the following picture of a tunnel, determined by a sequence of ceiling heights, namely, the sequence (7, 9, 6, 8, 7, 9, 8, 7, 9, 6, 8, 7, 9, 8, 7), and a corresponding sequence of floor heights, namely, the sequence (6, 2, 1, 5, 4, 3, 6, 6, 2, 1, 5, 4, 3, 6, 6). West of the tunnel, it is possible to see into the tunnel over a distance of 2 (units, whatever that is); this is depicted in blue. Inside the tunnel, it is possible to see into the tunnel over a maximum distance of 6; this is depicted in green. Note that on this picture, the heights of the blue and green segments are at the same level, but in general they could be at different levels. Also, there could be many green line segments, as there could be many parts of the tunnel where it is possible to see into the tunnel over the same maximum distance.



Write a program, stored in a file named `tunnel.py`, that performs the following task.

- The program prompts the user to input a file name. If there is no file with that name in the working directory, then the program outputs an error message and exits.
- The contents of the file should consist of exactly two lines with the same number of at last two (positive or negative) integers. Consecutive numbers are separated by at least one space, and there can be extra spaces, including empty lines, anywhere. Also, the i-th number of the first sequence should be strictly greater than the i-th number of the second sequence because at any point in the tunnel, the ceiling is above the floor. If that is not the case then the program outputs another error message and exits.
- The program then outputs:
  - the distance over which one can see into the tunnel when looking outside the tunnel from the West;
  - the maximum distance over which one can see into the tunnel when being inside the tunnel.

Here is a possible interaction:

```
$ cat tunnel_wrong_1.txt
7
1
$ python3 tunnel.py
Please enter the name of the file you want to get data from: tunnel_wrong_1.txt
Sorry, input file does not store valid data.
$ cat tunnel_wrong_2.txt
7 3
1 0  4
$ python3 tunnel.py
Please enter the name of the file you want to get data from: tunnel_wrong_2.txt
Sorry, input file does not store valid data.
$ cat tunnel_wrong_3.txt
7 3  9
1 3  4
$ python3 tunnel.py
Please enter the name of the file you want to get data from: tunnel_wrong_3.txt
Sorry, input file does not store valid data.
$ cat tunnel_wrong_4.txt
7 5  9

1 3  4

0 1  2
$ python3 tunnel.py
Please enter the name of the file you want to get data from: tunnel_wrong_4.txt
Sorry, input file does not store valid data.
$ cat tunnel_wrong_5.txt
7 5  9   B

1 3  4   A
$ python3 tunnel.py
Please enter the name of the file you want to get data from: tunnel_wrong_5.txt
Sorry, input file does not store valid data.
```

```
$ cat tunnel_1.txt
7 9 6 8
6 2 1 5
$ python3 tunnel.py
Please enter the name of the file you want to get data from: tunnel_1.txt
From the west, one can see into the tunnel over a distance of 2.
Inside the tunnel, one can see into the tunnel over a maximum distance of 3.
$ cat tunnel_2.txt


8 7 5 6 8 9 4 8 7

6 4 3 4 5 1 2 1 2

$ python3 tunnel.py
Please enter the name of the file you want to get data from: tunnel_2.txt
From the west, one can see into the tunnel over a distance of 2.
Inside the tunnel, one can see into the tunnel over a maximum distance of 4.
$ cat tunnel_3.txt
7 9 6 8 7 9 8 7 9 6 8 7 9 8 7
6 2 1 5 4 3 6 6 2 1 5 4 3 6 6
$ python3 tunnel.py
Please enter the name of the file you want to get data from: tunnel_3.txt
From the west, one can see into the tunnel over a distance of 2.
Inside the tunnel, one can see into the tunnel over a maximum distance of 6.
```