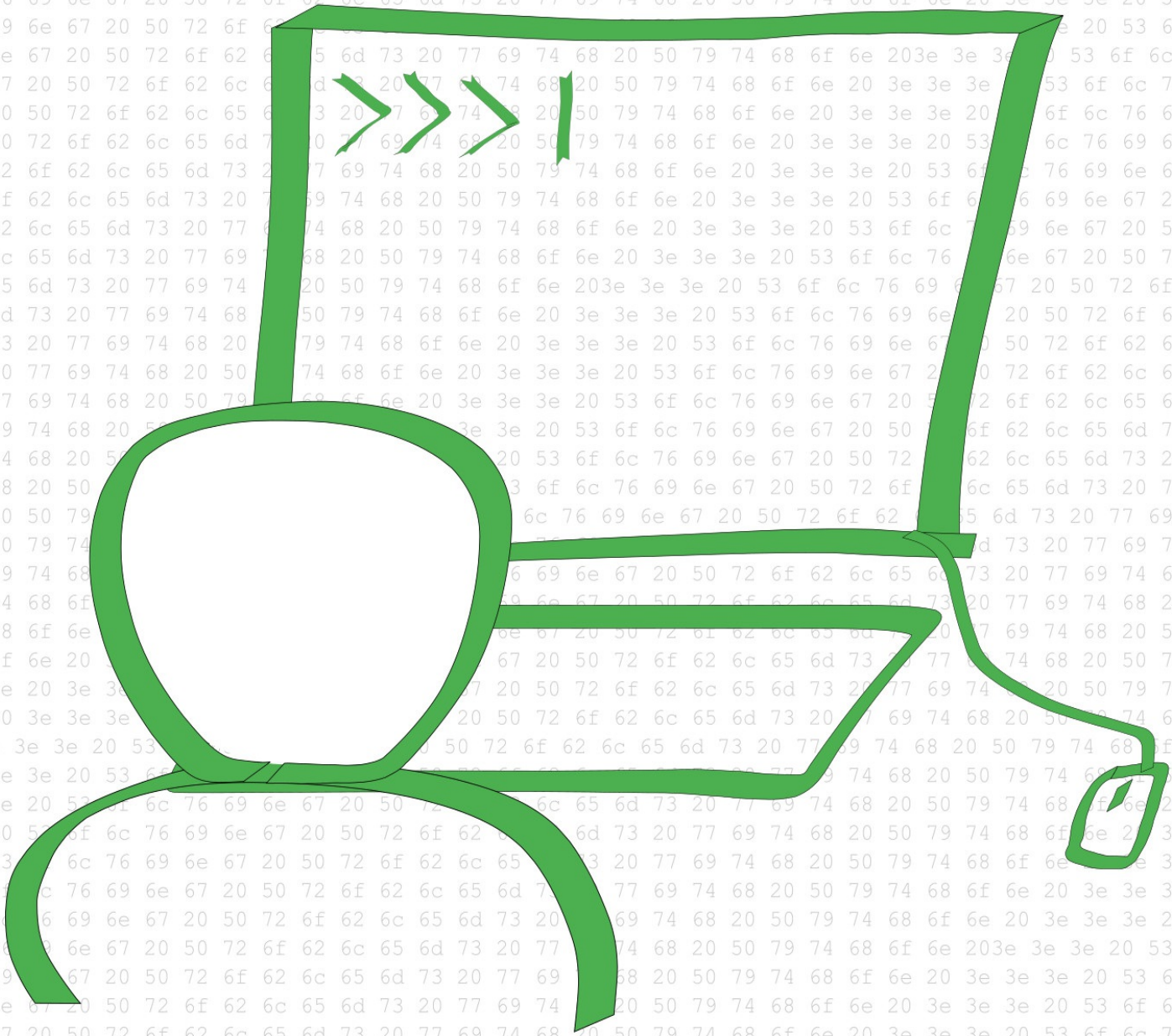
Solving Problems with Python

Peter Kučera, Jaroslav Výbošťok



www.creatingwithpython.com

Solving Problems with Python

Authors © Mgr. Peter Kučera, Mgr. Jaroslav Výbošťok

Design © Mgr. Peter Kučera

Translation: Jakub Ljutenko, Jana Ondičová, Joshua Ruggiero

First published, 2020

Version number: 20200511

Publisher: Peter Kučera

eBook preview

Visit the author's website at www.creatingwithpython.com

ISBN 978-80-570-1703-5 (pdf)

ISBN 978-80-570-1704-2 (epub)

ISBN 978-80-570-1705-9 (mobi)

Content

Introduction

Malcolm Gladwell says in his book *Outliers* that if you want to achieve success in a particular area, you have to endure 10,000 hours of practice in that field. The author uses this rule to show that talent alone is not enough, or even necessary, to achieve success. What is necessary, however, is a great amount of effort and systematic training. Some psychologists dismiss this idea.

Whether this claim is true or not, we believe that practice helps us get better in a particular area. We hope that these 64 exercises will help you improve your programming skills and gain the useful practice that will be appreciated not only by the *Maturita* committee, but also by your future self in your further studies or work.

If you have ever been so engulfed in programming that you forgot about the flow of time and everything around you, felt no hunger, thirst or exhaustion, you achieved flow. Psychologist Mihaly Csikszentmihalyi was the first to name and define flow.

An important prerequisite for achieving flow is that the level of difficulty of a given task be adequate to one's skills. When our skills develop at the same time as our tasks' level of difficulty rises, we start to feel passionate and get a sense of meaningfulness - also called experiencing flow.

It is not necessary for you to solve all of these exercises or solve them in their presented order. Choose your exercises in such a way that you enjoy solving them, ensuring that they pose a reasonable challenge for you, and you can enjoy programming :)

We wish you an enjoyable time solving these exercises and hopefully achieving flow and also amazing results at your *Maturita* exam.

Peter Kučera, Jaroslav Výbošťok

About this Collection of Exercises and the Informatics *Maturita* Exam

How does the *Maturita* exam work?

The informatics *Maturita* exam consists of an oral exam in front of a three-member *Maturita* committee. The student draws one of the approved *Maturita* tasks, which involves two exercises, one from programming and the second from the principles of informatics. The student gets a total of 30 minutes to prepare for the two exercises; the time the student spends on the individual exercises is completely up to the student. The preparation time is followed by a 20-minute oral exam.

In this e-book you will find exercises that meet the criteria for the first half of each of the *Maturita* tasks (the programming exercise).

Everything students are supposed to know for the *Maturita* exam is specified in the *Cieľové požiadavky na vedomosti a zručnosti maturantov z informatiky*, which you can find on the web page of the National Institute of Education (www.statpedu.sk).

What does this e-book include?

This e-book consists of 64 exercises (34 are graphic and 30 require the use of the console). Both the level of difficulty and the structure of these exercises fulfills the demands of the first part (exercise) of the informatics *Maturita* exam (algorithmic exercises). You can find these exercises in the first part of the book. At the end of every exercise, there is a link to its solution (the solution includes a link to the original exercise) which is located in the second part of the book. Solutions to some of the exercises include multiple possible methods of solving the particular exercise. Some exercises or solutions may also include additional questions that are supposed to lead you to more thinking and to help you develop your critical thinking and see new connections. Every solution ends with several symbols that define which areas of knowledge are used in the solution (e.g., text strings, lists, tuples, functions, etc.). This does not mean, however, that you will not be able to solve the exercise without using them.

At the beginning of the e-book, you can find a summary of the exercises along with the symbols for the approaches that may be necessary to solve them.

How to solve the exercises?

You can solve the exercises in any order. The order of the exercises in this e-book is completely random. Each exercise should take you about 20 to 30 minutes to solve (a combination of the time for preparation and the oral exam). During your *Maturita* exam, you do not have to fully solve the exercise in your preparation time, you can finalize it during the oral exam itself. The level of difficulty of the exercises in this e-book meets the approved level of difficulty for the first half of the *Maturita* tasks. We do not suggest that all of the exercises are of the same level in difficulty, however, they are all in the range meeting the *Maturita* criteria. You may find a particular exercise too hard and another too easy for you, but that may simply be caused by the fact that you are not used to the type or structure of the particular exercise or, on the contrary, that you have already solved many exercises like it before.

We recommend you keep track of your progress in tabs, where you can record the date you solved the exercise, the total time it took you as well as your own notes. From the log in your tracker, you should see that the time you need to solve each individual exercise is gradually shortening, meaning you are getting much-needed practice. It may also be useful to spend class time solving these exercises. You can have one shared tracker as a class (for example, using Google Drive) where every student keeps track of their own progress (in their own three columns). This way you can estimate the time needed to solve a particular exercise based on the time it took your classmates to do so or you can compare your times with your classmates.

Structure of the exercises

The first part of the *Maturita* task (the programming part) is supposed to have a clear objective the student is supposed to achieve by creating a program in a particular programming language. The means of achieving the objective are not supposed to be given to the student. It is the responsibility of the student to choose the means; whether the choice was the most effective and convenient is included in the assessment. This means that it should not be revealed whether the solution requires a loop, and if so which type of a loop, and also which type of a variable or data structure should be included in the neatest solution.

The exercises are structured in the form of bullet points representing smaller tasks (or particular attributes of the program). Often, the smaller tasks in the bullet points get gradually harder and represent the steps in which to solve the exercise. This structure is created to help even slightly less-skilled students at least partially solve the exercise. For instance, the first bullet point in an exercise in which a text file needs to be read may be to read and print the first line. By correctly executing this, the student shows that he or she can at least correctly upload a text file as input for the program and read its first line. However, the smaller task in the next bullet point may require reading and handling of the whole file. In some cases the first bullet point describes a smaller segment of the task in the next bullet point. In these cases, successful execution of the more difficult (second) task is sufficient and also shows that the student is capable of solving the basic (previous) task (which is designed to lead students who are unable to solve the more difficult task step by step to a solution).

Table of All Tasks

Table of Tasks with Graphics

Table of Tasks with Console

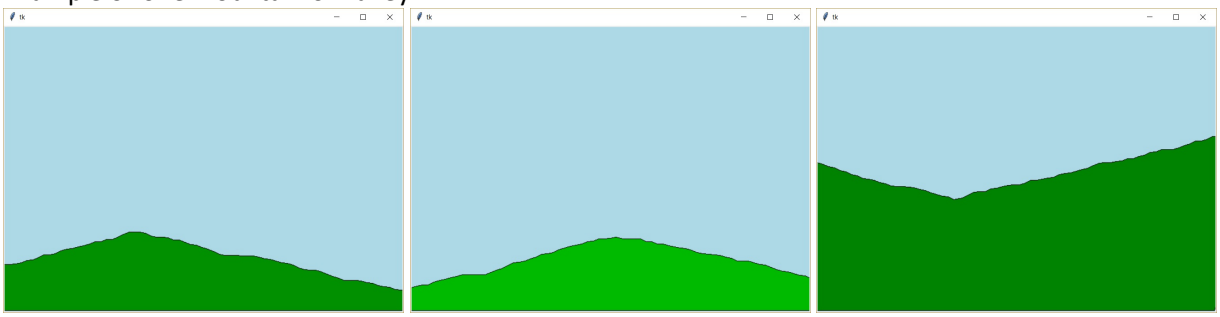
```
29. Image compression_____________________'', txt, fx()
32. File Conversion 1_____________________'', txt, fx()
34. Image Decompression___________________'', [], txt, fx()
37. Voting 2______________________________'', [], txt
38. Customer Satisfaction 1_______________'', [], txt
39. Customer Satisfaction 2_______________'', [], txt
40. Data Analysis_________________________'', [], txt
45. Vocabulary Learning___________________'', [], txt
46. Hangman_______________________________'', [], txt
48. Multiplication_____________________'', (), [], txt, fx()
51. Announcement Compression 1____________'', []
52. Announcement Compression 2____________'', []
63. Transit Research______________________(), [], txt
```

26 Landscape

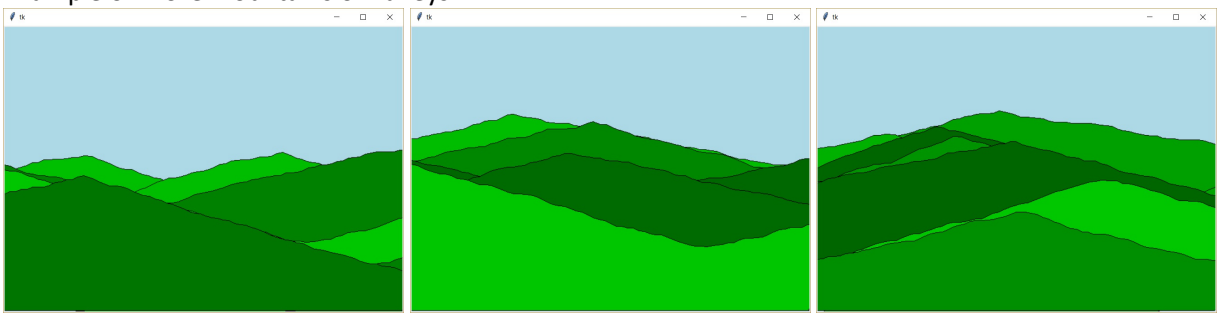
With this program we can generate and draw a random landscape. Create a program which:

- generates data for the mountain by randomly setting the x coordinate of the vertex and y coordinate of the mountain's initial height. For the mountain, it applies that the height before the vertex (the first part) does not decrease, and does not increase after the vertex. Changes in the mountain's surface can be made every 10 pixels and are random to the previous height level.
- draws the mountain using the command `canvas.create_polygon()`. The color of the mountain is a random tone of green.
- randomly decides whether to draw a mountain or a valley (whether it's first non-decreasing and then non-increasing or vice versa) and draws one mountain or one valley.
- repeatedly draws more random mountains / valleys which create the generated landscape,
- draws a new series of random mountains and valleys after pressing space.

Example of one mountain or valley:



Example of more mountains or valleys:



Questions:

1. Design a text file format for saving the data of the drawn and generated landscape.
2. Is it possible to estimate how many mountains (resp. valleys) were generated by the program? Justify your answer.

solution

40 Data Analysis

A local grocery store has decided that it wants to find out its customers' rate of satisfaction with the services it has provided. They have installed a box near the exit where customers can express their satisfaction / dissatisfaction on a touchscreen. All responses are written to a text file. Responses are contained in the text file `satisfaction_1.txt` (the files `satisfaction_2.txt` and `satisfaction_3.txt` are also at your disposal). Each line contains one customer response. The response contains the time of entry in the form of `hour:minute`, followed by one space and the text `yes` or `no` depending on whether the customer was satisfied or dissatisfied. The file contains responses from multimple days.

Example of the input text file:

```
15:38 yes
15:39 yes
14:33 yes
08:38 yes
07:42 yes
15:20 yes
```

Example of output:

```
1. day - number of reactions:2
2. day - number of reactions:1
3. day - number of reactions:1
4. day - number of reactions:2
Number of all responses: 6
Hour:7 Customer reactions:1
Hour:8 Customer reactions:1
Hour:14 Customer reactions:1
Hour:15 Customer reactions:3
Number of days: 4
```

Create a program which finds out and prints:

- the total number of responses,
- the total number of responses for the individual hours of the day, but only those hours when there where responses,
- the number of days during which responses were collected (let's assume that there was at least one response every day and that the input data is written in the same order as it was entered),
- the number of responses per individual days.

solution

26 Landscape

```
import tkinter, random
canvas = tkinter.Canvas(width=700, height=500, bg='lightblue')
canvas.pack()

def draw_hill():
    hill = []
    direction = random.choice((1,-1))                    #  {line A}
    hill.append(0)
    hill.append(random.randint(200, 500))
    summit = random.randint(100, 600)
    for i in range(summit // 10):
        new_value = hill[-1] + direction * random.randint(0, 5)
        hill.append(i*10+1)
        hill.append(new_value)
    direction = -1 * direction
    for i in range((700-summit) // 10 + 10):
        new_value = hill[-1] + direction * random.randint(0, 5)
        hill.append(i*10+summit)
        hill.append(new_value)

    hill = [0, 500] + hill + [700, 500]
    color = '#00{:02x}00'.format(random.randint(100, 200)) #  {line B}
    canvas.create_polygon(hill, fill=color, outline='black')

def draw(event):
    canvas.delete('all')
    for i in range(10):
        draw_hill()

#draw_hill()
canvas.bind_all('<space>', draw)
canvas.mainloop()
```

Questions:

1. What causes the notation in the line `{line A}`? How else could we write it?
2. Explain the notation in the line `{line B}`.
3. How is the change in elevation solved between the first part of the mountain and the second part?
4. How much information about the mountains is saved in memory while this program is running? Is it possible to optimize memory usage? If yes, how?

task
[] Cnv fx()

40 Data Analysis

```
opinions = [0] * 24

file = open('files/satisfaction_0.txt', 'r')
days = 0
time1 = '00:00'
cnt_in_day = 0
for line in file:
    line = line.strip()
    info = line.split()
    time2 = info[0]
    time2_split = info[0].split(':')
    hour = int(time2_split[0])
    minute = int(time2_split[1])
    opinions[hour] += 1
    if time2 < time1:
        days += 1
        print('{}. day - number of reactions:{}'.format(days, cnt_in_day))
        cnt_in_day = 1
    else:
        cnt_in_day += 1
    time1 = time2
print('{}. day - number of reactions:{}'.format(days, cnt_in_day))
total = sum(opinions)
print('Number of all responses:', total)
for i in range(24):
    if opinions[i] > 0:
        print('Hour:{} Costumer reactions:{}'.format(i, opinions[i]))
print('Days:', days)
```

Questions:

1. Based on what do we know that the recording is from the next day?
2. How does time comparison work?

task
'' [] txt