CM12003: Programming I: Coursework
Part 2: Python

Dr. Marina De Vos
Dept. of Computer Science
University of Bath
cssmdv@bath.ac.uk

Set:	20/10/2023 (week 6)Use the keyword key
Due :	19/01/2024 (week 16), 8pm
Percentage of overall unit mark:	50%
Submission Location:	Moodle
Submission Components:	code
Submission Format:	1 mastermind.zip file containing at least the file Mastermind.py
Anonymous Marking:	Y

1 Introduction

The coursework of this unit consists of two parts: Haskell assignment and a Python assignment This document provides the specification for the second part: the Python coursework. Please read it carefully.

You can use any IDE for the development of your scripts, but your scripts have to run when we use the command line on the university's Linux machines without requiring the installation of libraries, modules or other programmes.

Questions regarding the coursework can always be posted on the Moodle Forum or asked in the labs.

2 The Programme

2.1 Introduction

The main purpose of the assignment is to implement Mastermind as a computer game. The rules of the game are described in `https://magisterrex.files.wordpress.com/2014/07/mastermindrules.pdf` and `https://en.wikipedia.org/wiki/Mastermind_(board_`

`game)`. Your programme needs to be able to play with simulated human input with the guesses specified through the input file and implement a computer player that generates the guesses for the game. The assignment has specific functional requirements as described in the sections below. **Please read carefully.**

2.2 Functional Requirements

2.2.1 Programme call

We need to be able call your programme or script like this:

```
python Mastermind.py InputFile OutputFile [CodeLength] [MaximumGuesses] [AvailableColour]*
```

with

Mastermind.py : you have to call your main program Mastermind.py. We are running an automarker script.

InputFile : the file containing the input for the game (see Section 2.2.4)

OutputFile : the file that contains the output for the game (see Section 2.2.5)

[CodeLength]: Optional parameter indicating how long the code will be. The default is 5.

MaximumGuesses]: Optional parameter indicating how many attempts can be made in guessing the code. The default is 12.

[AvailableColours *]: Optional list of parameters indicating the available colours to select from. The defaults are red, blue, yellow, green and orange.

2.2.2 Human - Computer Version

The programme should cover two types of games: a simulation of a human playing the game (i.e. guesses are provided as part of the input file) or a computer player (i.e. your code will generate the guesses)

2.2.3 Exit Codes

Under no circumstance should your programme be allowed to crash. The programme should cover all eventualities. To indicate that something went wrong (e.g. I/O fault), you should communicate this by exit or return codes. You need to the following ones:

0 : The programme ran successfully

1 : Not enough programme arguments provided

2 : There was an issue with the input file

3 : There was an issue with the output file

4 : No or ill-formed code provided

5 : No or ill-formed player provided

2.2.4 Input File

The input file is expected to be formatted as follows:

- Line 1 specifies the code. Use the keyword ”code” followed by the colours of the pegs in the code
- Line 2 should specify ”human” or ”computer”.
- Line 3 onwards, if human play is chosen, should be guesses

Of course, there is no guarantee that the user (i.e read marker), is going to adhere to it. So, your programme has to be robust to deal with that. This is done through the exit codes (see Section 2.2.3) and messages to the output file (see Section 2.2.5).

An example of an input file (with some issues) is provided in Figure 1.

```
code red blue yellow
player human
red green blue
red red red
blue blue blue yellow
red green green
red blue yellow
red red red
```

Figure 1: An example input file

2.2.5 Output File

Whether we have a human or a computer player, the programme needs to log the game in the specified output file. If the input file provided no or ill-formed code, the output file will contain exactly one line:

```
No or ill-formed code provided
```

.

If the input file provided no or ill-formed player, the output file will contain exactly one line:

```
No or ill-formed player provided
```

.

Otherwise, you have for each guess until you have a correct guess or have reached the maximum number of guesses the following:

```
Guess X: [Blacks]* and [White]*
```

.

the keyword Guess followed by a count of the guess followed by ':' followed by "black" for each peg in the guess that matches the code and "white" for each peg in the guess that has the right colour but is in the wrong place. If the guess was incorrectly formatted, you write "Ill-formed guess provided"

If the code was broken, you write:

```
You won in X guesses. Congratulations!
```

with X the number of attempts that were needed.

When your input file contains more guesses, after the winning guess you should end your output with:

```
The game was completed. Further lines were ignored.
```

If the player was unable to guess the code, you write:

```
You lost. Please try again.
```

If they used up all their guesses, add to the output file:

```
You can only have X guesses.
```

with X the maximum number of guesses specified.

Figure 2 provides the generated output of game when the input specific in Figure 1 is passed on.

```
Guess 1: black white
Guess 2: black
Guess 3: Ill-formed guess provided
Guess 4: black
Guess 5: black black black
You won in 5 guesses. Congratulations!
The game was completed. Further lines were ignored.
```

Figure 2: An example input file

2.2.6 Computer Play

If the input file specifies that a computer player will play the game, the programme will produce the output file as specified in Section 2.2.5 based on the guesses of the computer. At the same time, an additional file, named computerGame.txt, is generated using the format for the input file as specified in Section 2.2.4 using the same code, a human player and the guesses of the computer player.

2.3 Non Functional Requirements

- your code needs to run on the University of Bath Linux machines (linux.bath.ac.uk)
- your code needs to be well documented
- you can use object-oriented programming concepts but there is absolutely no requirement.
- your code needs to be well-structured
- you are encouraged to use *git* to back-up your progress
- you are encouraged to use *test-driven development*, e.g. *pytest* to make sure your code works

3 Assessment

3.1 Conditions

The coursework will be conducted individually. Attention is drawn to the University rules on plagiarism and collusion. While software reuse (when allowed and with referencing the source) is permitted, we will only be able to assess your contribution. Please note that copying/reusing code from the internet could breach copyright or violate the software license. The use of software generators is not permitted for this coursework

3.2 Examples - Testing

A set of input and output examples, like the one in this document, will be provided for you to test your code with.

Make sure you create a range of test scenarios for your code so you can be certain that your code is working as expected. We will test its limits.

3.3 Marking Criteria

Marks are assigned in three categories: functionality, code structure, computer strategy.

- Functionality is assessed by a semi-automated process. A number of tests is run, and for recognised output, marks and feedback are produced automatically. Unrecognised output is marked manually by trying to fix **small** issues to make the tests run. We will attempt to fully debug your code.
- Code quality and computer strategy is assessed manually by the lecturers and senior tutors, by running and inspecting your code. A rubric with free comments will be used for providing feedback.

We will be using an automarker to test the functionality of your code. As part of this, we will run a script that calls your programme with a variety of values for the programme parameters. We will check the exit code and, if relevant, compare, using the Linux command *diff*, your output with the expected output. Be advised, the *diff* command is unforgiving. You need an exact match to get the marks.

For the top slice of the marking scheme, we assign marks on the sophistication of your computer player. The marks for complexity and the time spent on implementing the complexity are not linear. Do use your time wisely. Simple solutions will also gain marks.

Functionality (automarker):	Total:	40
	Exit Codes: Correct human player: Correct computer player:	6 28 6
Code structure:	Total:	30
	Presentation: Technique: Abstraction:	10 10 10
Computer Strategy	Total:	30
	fixed guess: random guess: more sophisticated algorithm:	max 2 max 5 30
Penalty	quick fixes	-10

Table 1: Mark breakdown

The requirements for code quality are as follows.

- Presentation: your code should be well laid out, with attention to whitespace, indentation, alignment, ordering of functions, commenting where necessary, and function and variable naming. Your style should be consistent.

- Technique: your code should use appropriate techniques for each situation, such as recursion, iteration, return values, breaks, continue. Your functions/methods should have a single and coherent purpose. The use of global variables should be justified. There should not be spurious imports or superfluous constructions (e.g. $b == True$ instead of b).

- Abstraction: your code should put commonly used functionality in a single function/method, to avoid repetition and aid readability.

Penalties: Your should not have no syntax or file name errors. We may correct minor errors for a penalty of up to -10 marks, and then mark as normal. Major errors may result in a functionality mark of zero.

In cases where it is not clear from the assignment how it should be marked, you may be called on to explain or demonstrate your program. Such cases include but are not limited to suspected plagiarism or collusion.

4 Submission Instructions

The deadline for this part of coursework is 8pm Friday 19 January 2024. Before the deadline upload your solution via Moodle (`https://moodle.bath.ac.uk/course/view.php?id=60461`). You need to submit a mastermind.zip file. The archive needs to contain all files needed for your coursework. It needs to contain at least the file Mastermind.py. This will be script are automarker will use. When unzipping, it should not create a directory. Before you upload your solution, make sure that the zip file contains all necessary files. Please download the file from Moodle and double check that you have uploaded the right file, with the content that you want to be marked. You are responsible for checking that you are submitting the correct material to the correct assignment.

5 Feedback

Individual detailed feedback will be provided via Moodle within three weeks of the submission deadline. Note that marks uploaded are provisional and need to be approved by the Board of Examiners for Units (BEU).