**Vidzeme University of Applied Sciences**

**Faculty of Engineering**

Introduction to Python and Data Exploration

**group 06**

**practical work #3**

Valmiera, 2024

**Table of Contents**

[1.0 Content 4](#_Toc162297565)

[2.0 Tasks and their solutions 5](#_Toc162297566)

[2.1 Task 1 5](#_Toc162297567)

[3.0 References 8](#_Toc162297568)

[4.0 Appendix (Code) 9](#_Toc162297569)

[4.1 Task\_3.1\_Gr.06.py 9](#_Toc162297570)

|  |  |  |  |
| --- | --- | --- | --- |
| Document versions | | | |
| Version | Status / Changes | Date | Author |
| 0.1 | First version implementation | 25. 03.2024 | Kristers Kalniņš |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Contacts and responsible (-s) | | | |
| Name Surname | Department | Position | Contact information (e-mail) |
| Emīls Konrāds | Group #06 | Coordinator | emils.konrads@va.lv |
| Gatis Jurisons | Group #06 | Member | gatis.jurisons@va.lv |
| Kristers Kalniņš | Group #06 | Member | kristers.kalnins@va.lv |
|  |  |  |  |
|  |  |  |  |

# Content

**Goal:** The team (further referenced as team 06) must complete the given task by the client (university professor) described in task 1 of unit 3.

**Content**: 1014 words, four images, three appendixes.

# Tasks and their solutions

Team 06 had organized a meeting after being given the task. The team had started organizing a workflow and work management chain the second Monday after being given the task due to the fact that the next lecture did not happen and the deadline for the works submission was a week later than usual. The team 06 decided to split the workload in 2 different categories:

* Documentation
* Coding

For the work flow we used the previously created Github repository. By the end of each week each team participant must send in a flow chain of how they see the app working, each demonstrating their plan of development.

## Task 1

The task was to write a code that would read a dataset file named “cereals.csv”. A file named “cereals\_dataset\_description.txt” was also provided, which contained the descriptions of the columns in the file “cereals.csv”. The specific information that had to be read from the file was: Name of the cereal, type of cereal and cereal rating. The task also required the code to contain the following:

* Identification of the lowest and highest value ratings of the cereal type.
* Calculation of the average cereal rating value for each type.

Two versions had to be implemented of the code: One with regular expressions and one without them. (Osis, Moodle.va.lv, 2024)

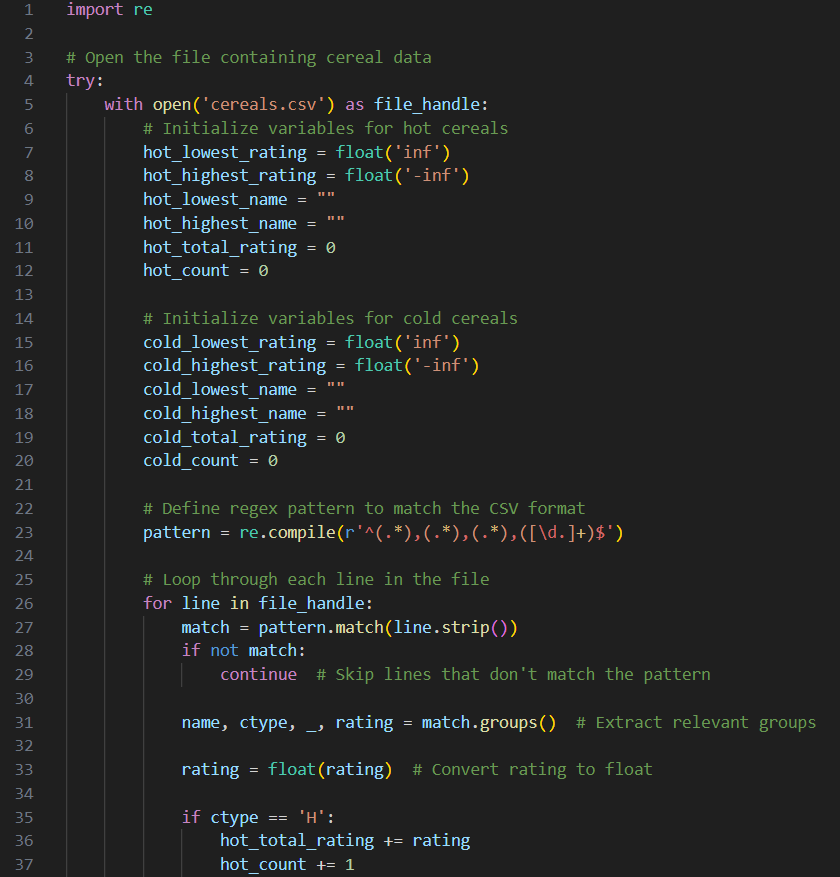


Image: 1.1. first part of RegEx code from (Task\_3.1\_Gr.06.py)

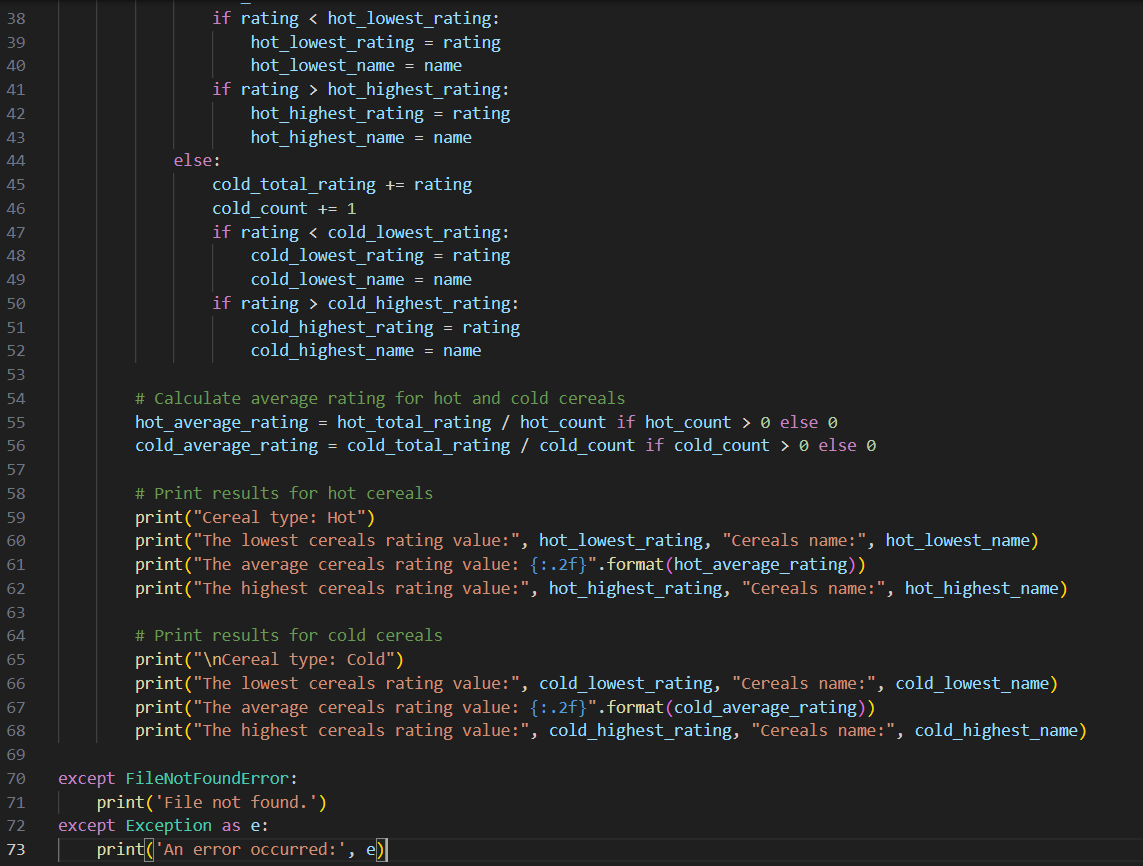


Image: 1.2. second part of RegEx code from (Task\_3.1\_Gr.06.py)

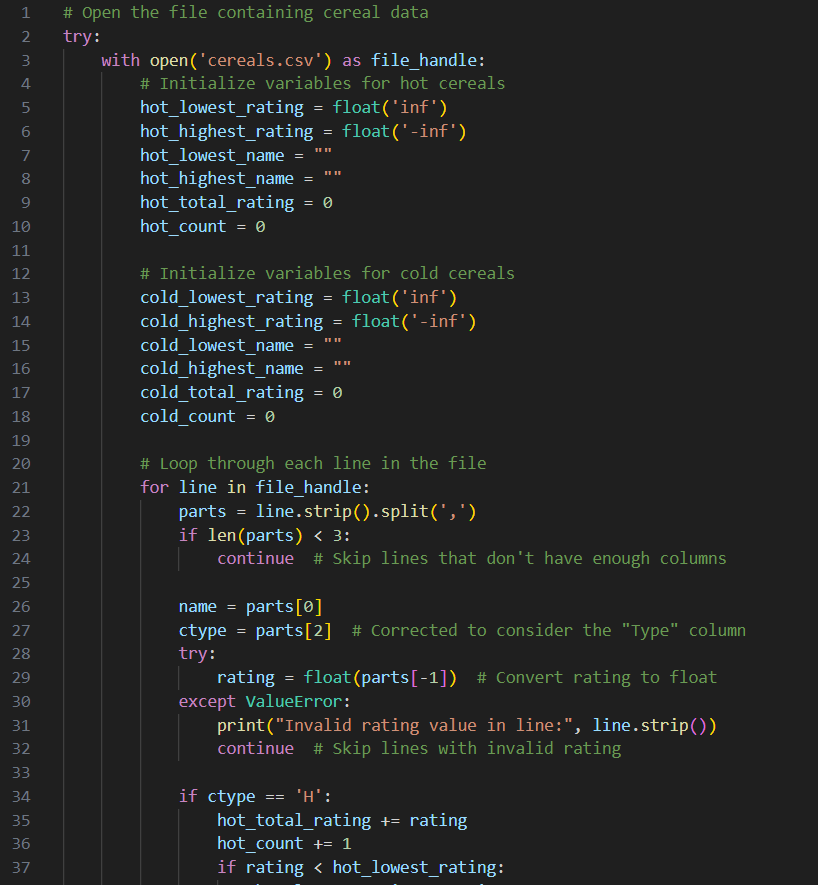


Image: 1.3. first part of no RegEx code from (Task\_3.1\_Gr.06.py)

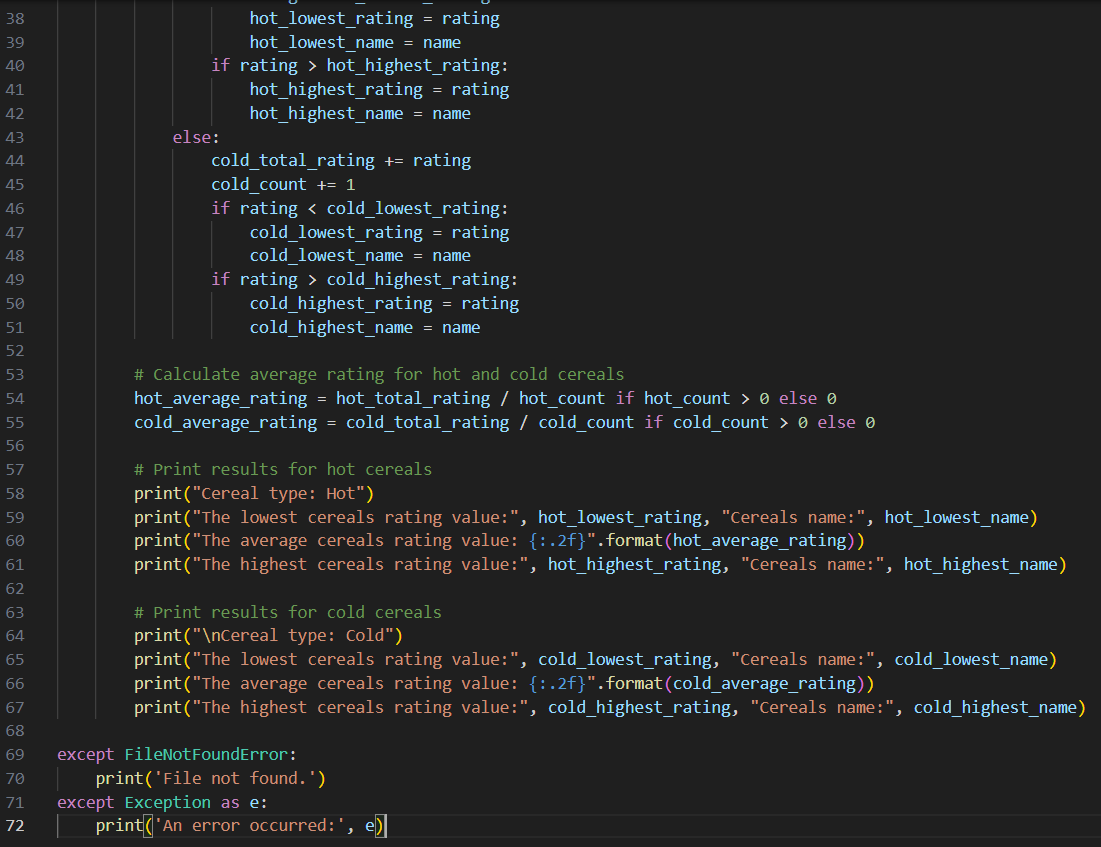


Image: 1.4. second part of no RegEx code from (Task\_3.1\_Gr.06.py)

# References

Osis, K. (2024, Mar). *Moodle.va.lv.* Retrieved from

https://moodle.va.lv/pluginfile.php/31526/mod\_resource/content/11/2024\_practical\_work\_3.pdf

Osis, K. (2024, Mar). *Moodle.va.lv.* Retrieved from Template of Document: https://moodle.va.lv/mod/resource/view.php?id=11925

# Appendix (Code)

## Task\_3.1\_Gr.06.py

Bez RegEx

# Open the file containing cereal data

try:

    with open('cereals.csv') as file\_handle:

        # Initialize variables for hot cereals

        hot\_lowest\_rating = float('inf')

        hot\_highest\_rating = float('-inf')

        hot\_lowest\_name = ""

        hot\_highest\_name = ""

        hot\_total\_rating = 0

        hot\_count = 0

        # Initialize variables for cold cereals

        cold\_lowest\_rating = float('inf')

        cold\_highest\_rating = float('-inf')

        cold\_lowest\_name = ""

        cold\_highest\_name = ""

        cold\_total\_rating = 0

        cold\_count = 0

        # Loop through each line in the file

        for line in file\_handle:

            parts = line.strip().split(',')

            if len(parts) < 3:

                continue  # Skip lines that don't have enough columns

            name = parts[0]

            ctype = parts[2]  # Corrected to consider the "Type" column

            try:

                rating = float(parts[-1])  # Convert rating to float

            except ValueError:

                print("Invalid rating value in line:", line.strip())

                continue  # Skip lines with invalid rating

            if ctype == 'H':

                hot\_total\_rating += rating

                hot\_count += 1

                if rating < hot\_lowest\_rating:

                    hot\_lowest\_rating = rating

                    hot\_lowest\_name = name

                if rating > hot\_highest\_rating:

                    hot\_highest\_rating = rating

                    hot\_highest\_name = name

            else:

                cold\_total\_rating += rating

                cold\_count += 1

                if rating < cold\_lowest\_rating:

                    cold\_lowest\_rating = rating

                    cold\_lowest\_name = name

                if rating > cold\_highest\_rating:

                    cold\_highest\_rating = rating

                    cold\_highest\_name = name

        # Calculate average rating for hot and cold cereals

        hot\_average\_rating = hot\_total\_rating / hot\_count if hot\_count > 0 else 0

        cold\_average\_rating = cold\_total\_rating / cold\_count if cold\_count > 0 else 0

        # Print results for hot cereals

        print("Cereal type: Hot")

        print("The lowest cereals rating value:", hot\_lowest\_rating, "Cereals name:", hot\_lowest\_name)

        print("The average cereals rating value: {:.2f}".format(hot\_average\_rating))

        print("The highest cereals rating value:", hot\_highest\_rating, "Cereals name:", hot\_highest\_name)

        # Print results for cold cereals

        print("\nCereal type: Cold")

        print("The lowest cereals rating value:", cold\_lowest\_rating, "Cereals name:", cold\_lowest\_name)

        print("The average cereals rating value: {:.2f}".format(cold\_average\_rating))

        print("The highest cereals rating value:", cold\_highest\_rating, "Cereals name:", cold\_highest\_name)

except FileNotFoundError:

    print('File not found.')

except Exception as e:

    print('An error occurred:', e)

Ar RegEx

import re

# Open the file containing cereal data

try:

    with open('cereals.csv') as file\_handle:

        # Initialize variables for hot cereals

        hot\_lowest\_rating = float('inf')

        hot\_highest\_rating = float('-inf')

        hot\_lowest\_name = ""

        hot\_highest\_name = ""

        hot\_total\_rating = 0

        hot\_count = 0

        # Initialize variables for cold cereals

        cold\_lowest\_rating = float('inf')

        cold\_highest\_rating = float('-inf')

        cold\_lowest\_name = ""

        cold\_highest\_name = ""

        cold\_total\_rating = 0

        cold\_count = 0

        # Define regex pattern to match the CSV format

        pattern = re.compile(r'^(.\*),(.\*),(.\*),([\d.]+)$')

        # Loop through each line in the file

        for line in file\_handle:

            match = pattern.match(line.strip())

            if not match:

                continue  # Skip lines that don't match the pattern

            name, ctype, \_, rating = match.groups()  # Extract relevant groups

            rating = float(rating)  # Convert rating to float

            if ctype == 'H':

                hot\_total\_rating += rating

                hot\_count += 1

                if rating < hot\_lowest\_rating:

                    hot\_lowest\_rating = rating

                    hot\_lowest\_name = name

                if rating > hot\_highest\_rating:

                    hot\_highest\_rating = rating

                    hot\_highest\_name = name

            else:

                cold\_total\_rating += rating

                cold\_count += 1

                if rating < cold\_lowest\_rating:

                    cold\_lowest\_rating = rating

                    cold\_lowest\_name = name

                if rating > cold\_highest\_rating:

                    cold\_highest\_rating = rating

                    cold\_highest\_name = name

        # Calculate average rating for hot and cold cereals

        hot\_average\_rating = hot\_total\_rating / hot\_count if hot\_count > 0 else 0

        cold\_average\_rating = cold\_total\_rating / cold\_count if cold\_count > 0 else 0

        # Print results for hot cereals

        print("Cereal type: Hot")

        print("The lowest cereals rating value:", hot\_lowest\_rating, "Cereals name:", hot\_lowest\_name)

        print("The average cereals rating value: {:.2f}".format(hot\_average\_rating))

        print("The highest cereals rating value:", hot\_highest\_rating, "Cereals name:", hot\_highest\_name)

        # Print results for cold cereals

        print("\nCereal type: Cold")

        print("The lowest cereals rating value:", cold\_lowest\_rating, "Cereals name:", cold\_lowest\_name)

        print("The average cereals rating value: {:.2f}".format(cold\_average\_rating))

        print("The highest cereals rating value:", cold\_highest\_rating, "Cereals name:", cold\_highest\_name)

except FileNotFoundError:

    print('File not found.')

except Exception as e:

    print('An error occurred:', e)