**Vidzeme University of Applied Sciences**

**Faculty of Engineering**

Introduction to Python and Data Exploration

**group 06**

**practical work #5**

Valmiera, 2024

**Table of Contents**

[Content 4](#_Toc165964444)

[Tasks and their solutions 5](#_Toc165964445)

[Task 1 5](#_Toc165964446)

[Task 2 6](#_Toc165964447)

[Task 3 7](#_Toc165964448)

[References 8](#_Toc165964449)

[Appendix (Code) 9](#_Toc165964450)

[Python\_1\_Gr.06.py 9](#_Toc165964451)

[Python\_2\_Gr.06.py 10](#_Toc165964452)

[Python\_3\_Gr.06.py 11](#_Toc165964453)

|  |  |  |  |
| --- | --- | --- | --- |
| Document versions | | | |
| Version | Status / Changes | Date | Author |
| 0.1 | First version implementation | 05. 07.2024 | Emīls Konrāds |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 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 5.

**Content**: two charts, words, images, appendixes.

# Tasks and their solutions

Team 06 organized a meeting after the given the task, over the weekend of the following weeks, after discussing work roles and responsibilities. The team 06 decided to split the workload in 2 different categories:

* Documentation
* Coding

While all participants take part in all parts of the workload, some team members take more of a lead in documenting and only help in the coding process. For the work flow a GitHub repository was updated with a new folder and maintained by the team coordinator.

## Task 1

After following the tasks requirements, the basic requirements where sett: (K.Osis, 2024)

* Use URL.
* Split URL into component parts.
* Add Error checks.
* Count characters and display them till the count of 1700 is reached.

The following is a image of the code that Group 6 created to do the main subtasks of the homework project. Importing a socket (which is a module that provides socket operations and some related functions for them). After the Input of the URL, we try and split it and divide it into parts. Next up is the creation of a socket connection to the host of the URL, simply put like a book a connection needs to be opened, read, and then closed and these functions are executed in the code.

The last part of the task is executed a simple counter that works on a bool base that counts and displays the characters till it reaches a count of 1700.  
  
A screenshot of a computer program

Description automatically generated

Image: 1.1. Code from (Task\_5.1\_Gr.06.py)

## Task 2

After following the tasks requirements, the basic requirements where sett: (K.Osis, 2024)

* Use Beautiful-Soup methodology to read specific parts of the page.
* Establish a connection to a movie database webpage.
* Obtain the following information from the page:
  + - The writer
    - Actors
    - Stars
    - Number of awards
    - Cast list.

To achieve the tasks successfully completion the codes followed the pattern that the description of the task set. Importing requests and BS4 (Python softwhere foundation, 2024) (Basically a preset function of code that goes through reads the HTML of the page and lets us take specific elements using the inspector view. Hence the name Soup since we are swimming trough it to find what we are looking for). Placing the URL and mimicking the browser we can achieve the BS4 method to go trough the HTML form the URL. We obtain the content of the page and start to call out classes to get information about these classes. Straight forward and simple.

A screen shot of a computer program

Description automatically generated

Image: 1.2. Code from (Task\_5.2\_Gr.06.py)

## Task 3

After following the tasks requirements, the basic requirements where sett: (K.Osis, 2024)

* Use OpenStreetMap API.
* Read User Latitude and longitude.
* Use JSON and the API to format and display:
  + Location.
  + House Nr.
  + Street name.
  + City.
  + Postal Code.
  + Country.
  + Country Code.

Following the basic start that was covered in the previous tasks. After we request the URL of the page imputing our desired Lan and Len trough the code. Then check if the request is successful and work format the code to read the JSON format for further workload.

Simply put we get all the information of the location using a simple method to check if the required data is available for the Place\_Name, Plance\_Type, Adress, House\_Nr., Street\_Name, City, Postcode, Country, etc. After we add a display which works as a simple print. Then at the end we call out the main functions which are: Entering your Latitude and Longitude.

A screen shot of a computer program

Description automatically generated

Image: 1.2. Code from (Task\_5.3\_Gr.06.py)

# References

K.Osis. (2024, 04 16). *Moodle.lv*. Retrieved from https://moodle.va.lv/pluginfile.php/31936/mod\_resource/content/7/2024\_practical\_work\_5.pdf

Python softwhere foundation. (2024). *pypi.org*. Retrieved from https://pypi.org/project/beautifulsoup4/

# Appendix (Code)

## Python\_5.1\_Gr.06.py

import socket

# Prompt the user to input a URL

url = input("Enter the URL: ")

try:

    # Split the URL into its component parts

    parts = url.split("/")

    host = parts[2]

    # Create a socket connection to the host

    sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

    sock.connect((host, 80))

    # GET request to retrieve the URL

    request = f"GET {url} HTTP/1.1\r\nHost: {host}\r\n\r\n"

    sock.sendall(request.encode())

    # Receive and count the characters in the response

    characters = 0

    while True:

        data = sock.recv(1024)

        if not data:

            break

        characters += len(data)

        if characters > 1700: #If bigger than 1700

            print(f"\nStopping display at 1700 characters\n")

            break

        print(data.decode(), end="")

except Exception as e:

    # Handle any exceptions that occur during execution

    print(f"Error: {e}")

finally:

    # Close

    sock.close()

## Python\_5.2\_Gr.06.py

import requests

from bs4 import BeautifulSoup

# URL of the IMDb page to scrape

URL = 'https://www.imdb.com/title/tt6084202/'

# Here we basicly mimic the URL trying to make the Browser think we are Legit

headers = {

    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/124.0.0.0 Safari/537.36'

}

# GET request to the URL and parsing the HTML content

page = requests.get(URL, headers=headers)

soup = BeautifulSoup(page.content, 'html.parser')

# Director from the page

director\_section = soup.find('a', class\_='ipc-metadata-list-item\_\_list-content-item ipc-metadata-list-item\_\_list-content-item--link', href=True)

if director\_section:

    director = director\_section.text.strip()

# Writers from the page

writter\_section = soup.find\_all('a', class\_='ipc-metadata-list-item\_\_list-content-item ipc-metadata-list-item\_\_list-content-item--link', )

if writter\_section:

    writter = writter\_section[1].text.strip()

    writter1 = writter\_section[2].text.strip()

# Actor if it's mentioned in the actor section

actor\_section = soup.find('div', class\_='sc-bfec09a1-7 gWwKlt')

if actor\_section:

    actors = actor\_section.find\_all('a')

    for actor in actors:

        if "Oto Brantevics" in actor.text:

            actor\_name = actor.text.strip()

# Awards and nominations

award\_section = soup.find('span', class\_="ipc-metadata-list-item\_\_list-content-item")

for award in award\_section:

    if "9 wins" in award.text:

        wins\_nominations = award.text.strip()

print(f"Director: {director}")

print(f"Writters: {writter}, {writter1}")

print(f'Actor: {actor\_name}')

print(f'Award: {wins\_nominations}')

## Python\_5.3\_Gr.06.py

import requests

def reverse\_geocode(latitude, longitude):

    # Basicly Creates the URL that we use for the apps workflow

    url = f"https://nominatim.openstreetmap.org/reverse?lat={latitude}&lon={longitude}&format=json"

    # Send a GET request to the API

    response = requests.get(url)

    # Check if the request was successful (the statuse code is 200)

    if response.status\_code == 200:

        data = response.json()

        # Extract relevant information from the response

        place\_name = data.get('display\_name', 'Not available')

        place\_type = data.get('type', 'Not available')

        address = data.get('address', {})

        house\_number = address.get('house\_number', 'Not available')

        street\_name = address.get('road', 'Not available')

        city = address.get('city', 'Not available')

        postcode = address.get('postcode', 'Not available')

        country = address.get('country', 'Not available')

        country\_code = address.get('country\_code', 'Not available')

        # Display the extracted information

        print("Place Name:", place\_name)

        print("Place Type:", place\_type)

        print("House Number:", house\_number)

        print("Street Name:", street\_name)

        print("City:", city)

        print("Postcode:", postcode)

        print("Country:", country)

        print("Country Code:", country\_code)

    else:

        # Basic error to check if we got the datta

        print("Failed to retrieve geocoding data.")

    # basic Inputs

latitude = input("Enter latitude: ")

longitude = input("Enter longitude: ")

# Call the function and get Lan and Lon

reverse\_geocode(latitude, longitude)