Web Scraping

Web Scraping je proces pridobivanja informacij z interneta.

S pomočjo web scrapinga lahko napišemo skripto, ki nas opozori, ko se nam približuje slabo vreme. Napišemo lahko skripto, ki nam pridobi vse tweete specifične osebe, pridobi trenutne informacije o stanju na cestah. Napišemo lahko skripto, ki se sprehodi čez članke na wikipediji in izpiše vse stavke, ki vsebujejo iskane besede, ipd.

Ponavadi ljudje uporabljamo internet preko **HTTP (HyperText Transfer Protocol)**.

> (v grobem): V browser napišemo spletni naslov katerega želimo obiskati. Browser nato izvede klic za pridobitev te spletne strani. Če spletna stran obstaja je posredovana nazaj v browser in ta nam prikaže spletno stran.

Za uporabo HTTP v pythonu obstaja knjižnjica **requests**.

> Dokumentacija: https://docs.python-requests.org/en/master/ (https://docs.python-requests.org/en/master/)

To je 3rd party knjižnjica, kar pomeni, da ne pride avtomatično z inštalacijo pythona. Zato jo moramo sami inštalirati.

Za inštalacijo 3rd party knjižnjic zapišemo ukaz **pip install <knjižnjica>** v terminal:

```
pip install requests
```

In [ ]:

```
import requests
```

Za začetek bomo ustvarili preprosti **GET** Request.

S tem tem requestom želimo pridobiti podatke iz določenega vira.

V našem primeru bomo pridobili podatke s sledečega URL: https://api.github.com (https://api.github.com)

In [ ]:

```
url = "https://api.github.com"
response = requests.get(url)

print(response)
```

Response object

Naš **get()** klic nam vrne *response object* znotraj katerega imamo informacije in podatke glede našega klica.

STATUS CODE

https://en.wikipedia.org/wiki/List_of_HTTP_status_codes (https://en.wikipedia.org/wiki/List_of_HTTP_status_codes)

Prva informacija našega klica, ki bi nas zanimala je **status code**.

V osnovi se kode delijo na:

- 1xx - informational response - request je biu prejet. Nadaljujemo s procesom
- 2xx - successful - request je bil uspešno prejet, razumljen in sprejet
- 3xx - redirection - dodatne akcije so potrebne za dokončanje requesta
- 4xx - client error - request vsebuje slabo syntaxo oziroma ne more biti izpolnjen
- 5xx - server error - request je bil pravilen vendar server ne more dokončati requesta

In [ ]:

```
url = "https://api.github.com"
response = requests.get(url)

print(response.status_code)
```

Naš **get()** klic bi moral vrniti status kodo 200, kar pomeni da ni prišlo do napak in se je klic uspešno izvršil.

In [ ]:

```
url = "https://api.github.com/ne_obstaja"
response = requests.get(url)

print(response.status_code)
```

Če želimo prodibiti neke podatke, ki ne obstajajo, bomo dobili kodo **404 - Not Found**.

S pomočjo response_code lahko nadzorujemo logiko našega programa:

In [ ]:

```
url = "https://api.github.com" # 200
#url = "https://api.github.com/ne_obstaja" # 404
response = requests.get(url)

if response.status_code == 200:
    print("Uspešen GET Request")
    print("Nadaljuj z obdelavo podatkov")
elif response.status_code == 404:
    print("Error! Te podatki ne obstajajo.")
```

Namest primerjanja status code lahko uporabimo kar response object. Response object ima vrednost True, če je response code med 200 in 400. V nasprotnem primeru ima vrednost False.

In [ ]:

```
url = "https://api.github.com" # 200
#url = "https://api.github.com/ne_obstaja" # 404
response = requests.get(url)

if response:
    print("Uspešen GET Request")
else:
    print("Error!")
```

Tak način preverjanja naj se uporablja le, če želimo preveriti ali je bil request uspešno sprocesiran ali ne.

204 je status koda, ki nam pove, da je bil request uspešno sprocesiran vendar ni nobenih podatkov za vrniti.

PODATKI

Podatke do katerih smo hoteli dostopati - **payload** - imamo shranjenje v telesu našega sporočila - **message body**.

Do njih lahko dostopamo v različnih oblikah:

in bytes:

In [ ]:

```
url = "https://api.github.com"
response = requests.get(url)

print(type(response.content))
print(response.content)
```

Podatke ponavadi hočemo v obliki **stringa**. Do njih lahko dostopamo na sledeč način:

In [ ]:

```
url = "https://api.github.com"
response = requests.get(url)

print(type(response.text))
print(response.text)
```

Za pretvorbo v tekst se potreuje **encoding scheme**. Requests knjižnjica poizkuša uganiti encoding shemo glede na response headerje. Lahko po encoding shemo podamo explicitno:

In [ ]:

```
url = "https://api.github.com"
response = requests.get(url)

response.encoding = "utf-8"

print(response.encoding)
print(response.text)
```

Če si pogledamo naše pridobljene podatke vidimo, da so v obliki JSON. Da jih pridobimo v taki obliki imamo znotraj knjiznjice prirocno metodo:

In [ ]:

```
url = "https://api.github.com"
response = requests.get(url)
data = response.json()

print(type(data))
print(data)
```

Če želimo izvedeti kaj več o informacij, kot so metapodatki našega odgovora, potrebujemo pogledati v **headers** odgovora.

Headers

Headers držijo informacije kot so, content type naših podatkov ali kako dolgo naj imamo odgovor chached.

Do headerjev dostopamo preko **.headers** atributa, ki nam vrne headers v dictionary obliki.

In [ ]:

```
url = "https://api.github.com"
response = requests.get(url)

print(response.headers)
```

Če želimo videti kakšnega datatipa so vrnjeni podatki lahko dostopamo do **Content-Type** headerja.

HTTP specifikacije določajo, da so headerji *case insensitive* kar pomeni, da velike črke ne vplivajo na klicanje naših headerjev.

In [ ]:

```
url = "https://api.github.com"
response = requests.get(url)

print(response.headers["Content-Type"])
print(response.headers["content-type"])
```

In [ ]:

```
```

Za bolj realen primer bomo pridobili podatke o praznikih in dela prostih dneh v Republiki Sloveniji.

Informacije o podatkih lahko najdemo na sledeči spletni strani: https://podatki.gov.si/dataset/seznam-praznikov-in-dela-prostih-dni-v-republiki-sloveniji/resource/eb8b25ea-5c00-4817-a670-26e1023677c6 (https://podatki.gov.si/dataset/seznam-praznikov-in-dela-prostih-dni-v-republiki-sloveniji/resource/eb8b25ea-5c00-4817-a670-26e1023677c6)

Na spletni strani vidimo, da so podatki shranjeni v **csv** formatu.

Imajo 8 stolpcev:

- id
- Datum
- Ime praznika
- Dan v tednu
- Dela prost dan
- Dan
- Mesec
- Leto

Dejanske podatke lahko pridobimo na URL: https://podatki.gov.si/dataset/ada88e06-14a2-49c4-8748-3311822e3585/resource/eb8b25ea-5c00-4817-a670-26e1023677c6/download/seznampraznikovindelaprostihdni20002030.csv (https://podatki.gov.si/dataset/ada88e06-14a2-49c4-8748-3311822e3585/resource/eb8b25ea-5c00-4817-a670-26e1023677c6/download/seznampraznikovindelaprostihdni20002030.csv)

In [ ]:

```
import requests

url = "https://podatki.gov.si/dataset/ada88e06-14a2-49c4-8748-3311822e3585/resource

response = requests.get(url)
#print(r.encoding)
response.encoding = "utf-8" # treba dodat, ker če ne maš ISO-8859-1 kar pa ne prepo

data = response.text
print(data)
```

Naša naloga bi sedaj lahko bila, da preverimo koliko praznikov pade na določen dan v tednu, za leto 2022.

In [ ]:

```
import requests

url = "https://podatki.gov.si/dataset/ada88e06-14a2-49c4-8748-3311822e3585/resource

response = requests.get(url)
#print(r.encoding)
response.encoding = "utf-8" # treba dodat, ker če ne maš ISO-8859-1 kar pa ne prepo

data = response.text

rezultat = {}
for vrstica in data.split("\r\n"):
    v_splitted = vrstica.split(";")
    #print(v_splitted)
    if v_splitted[-1] == "2022":
        print(v_splitted)
        dan = v_splitted[2]
        if dan in rezultat.keys():
            rezultat[dan] += 1
        else:
            rezultat[dan] = 1

print("Število praznikov na specifični dan: ")
print(rezultat)
```

URL katerega smo uporabili predstavlja API portala OPSI.

API (**Application Programming Interface**) predstavlja povezavo med dvema računalnikoma oziroma programoma.

V našem primeru je naš program kontaktiral portal OPSI preko API in pridobil podatke.

Veliko spletnih strani ima vzpostavljene API. Preko njihovih specifičnih URL-jev lahko tako dostopamo do njihovih urejenih podatkov.

Formati takšnih podatkov so velikokrat standardni, kot so CSV, XML, JSON, itd...

Za primer bolj naprednega API si poglejmo **coingecko.com**. To je spletna platforma za spremljanje trgovanja s kriptovalutami. Imajo informacije o trenutni ceni, volumnu, market cap, novicah, itd.

> https://www.coingecko.com/en (https://www.coingecko.com/en)

Dokumentacijo svojega API imajo lepo zapisano na:

> https://www.coingecko.com/api/documentations/v3#/ (https://www.coingecko.com/api/documentations/v3#/)

Vidimo, da so vse metode **GET** in okvirno kako so URL sestavljeni.

Za primer vzemimo nalogo, kjer moramo poiskati trenutno ceno Bitcoina v €.

API kateri nam bi lahko rešil nalogo je **GET /simple/price**. Če ga odpremo vidimo, da lahko izberamo še dodatne parametre in, da nam spletna stran sama zgenerira URL in nam tudi nudi možnost testiranja tega URL.

> https://api.coingecko.com/api/v3/simple/price?ids=bitcoin&vs_currencies=eur (https://api.coingecko.com/api/v3/simple/price?ids=bitcoin&vs_currencies=eur)

Podatke bomo dobili vrnjene v JSON formatu. JSON format je podoben python dictionary.

Če sedaj odpremo podani URL se nam v brskalniku izpišejo JSON podatki katere bi prejeli, če bi URL klicali s programom.

In [ ]:

```
import requests

url = "https://api.coingecko.com/api/v3/simple/price?ids=bitcoin&vs_currencies=eur"

r = requests.get(url)
data = r.json()
print(data)
print("Cena BTC v €: ", data["bitcoin"]["eur"])
```

Če bi sedaj podatke želeli v $ namesto v €, bi morali spremeniti URL.

In [ ]:

```
import requests

url = "https://api.coingecko.com/api/v3/simple/price?ids=bitcoin&vs_currencies=usd"

r = requests.get(url)
data = r.json()
print(data)
print("Cena BTC v $: ", data["bitcoin"]["usd"])
```

URL je v grobem sestavljen iz:

- **Base URL**, ki predstavlja pot do spletne strani. `api.coingecko.com/api/v3/simple/price`
- **Query parameters**, ki predstavljajo parametre katere lahko spreminjamo. Pričnejo se po `?`

Query parameters so sestavljeni iz:

- **imena parametra** - `id`
- **=**, enačaja
- **vrednosti parametra** - `bitcoin`

Med seboj so parametri ločeni z `&`.

Recimo, da imamo naš portfolio sestavljen iz sledečih kriptovalit:

```
["bitcoin", "ethereum", "cardano", "polkadot", "secret"]
```

Ko zaženemo naš program bi radi, da nam izpiše trenutno ceno vsakega kovanca v našem portfoliju. To pomeni, da bomo morali URL-je dinamično kreirati.

In [ ]:

```
import requests

my_portfolio = ["bitcoin", "ethereum", "cardano", "polkadot", "secret"]

for coin in my_portfolio:
    url = f"https://api.coingecko.com/api/v3/simple/price?ids={coin}&vs_currencies=

    r = requests.get(url)
    data = r.json()
    print(f"Cena {coin} v €: ", data[coin]["eur"])
```

In [ ]:

```

```

Oziroma, namesto, da dinamično spreminjamo URL lahko naše query parametre definiramo v get() metodi.

In [ ]:

```
import requests

my_portfolio = ["bitcoin", "ethereum", "cardano", "polkadot", "secret"]

for coin in my_portfolio:
    url = f"https://api.coingecko.com/api/v3/simple/price"

    r = requests.get(url, params={"ids": coin, "vs_currencies":"eur"})
    data = r.json()
    print(f"Cena {coin} v €: ", data[coin]["eur"])
```

In [ ]:

```

```

Naloga:

Pridobite **daily** podatke o **ceni in market_cap** za do 3 dni nazaj za naš portfolijo. Podatki naj bodo v €.

```
["bitcoin", "ethereum", "cardano", "polkadot", "secret"]

OUTPUT

bitcoin
Price in €: 50120.35,      MC: 946129966385.45
Price in €: 51792.97,      MC: 977748021681.97
Price in €: 53231.48,      MC: 1004952689342.15

ethereum
Price in €: 3512.04,      MC: 415518933689.03
Price in €: 3825.25,      MC: 451060688164.44
Price in €: 3930.32,      MC: 464722167457.57

cardano
Price in €: 1.57,      MC: 50210489214.95
Price in €: 1.66,      MC: 53018653910.54
Price in €: 1.71,      MC: 54707905428.96

polkadot
Price in €: 34.24,      MC: 36178751146.04
Price in €: 36.70,      MC: 38654769090.58
Price in €: 37.37,      MC: 39428742405.97

secret
Price in €: 6.22,      MC: 926683065.11
Price in €: 6.46,      MC: 961627834.47
Price in €: 6.38,      MC: 951412154.96:
```

In [ ]:

```
import requests
my_portfolio = ["bitcoin", "ethereum", "cardano", "polkadot", "secret"]

for coin in my_portfolio:
    url = f"https://api.coingecko.com/api/v3/coins/{coin}/market_chart?vs_currency=

    r = requests.get(url)

    data = r.json()
    print(coin)
    for i in range(3):
        print(f"Price in €: {data['prices'][i][1]:.2f}, \t MC: {data['market_caps']
    print()
```

In [ ]:

```

```

Naloga:

S pomočjo webscrapinga preverite ali bi se lahko z Bicikelj odpeljali domov.

Vaša začetna postaja je TRG MDB

Vaša končna postaja je STARA CERKEV.

Preverite ali je na začetni postaji vsaj 1 prosto kolo in ali je na končni postaji vsaj 1 prosto parkirno mesto.

Podatke lahko dobite na sledečem linku v JSON formatu. Podatki o prostih mestih in kolesih se nahaja v "station" delu.

```
free nam pove koliko prostih mest je na postaji.
available nam pove koliko koles je prostih za izposojo.
```

https://opendata.si/promet/bicikelj/list/ (https://opendata.si/promet/bicikelj/list/)

In [ ]:

```
# Rešitev:

import requests

url = "https://opendata.si/promet/bicikelj/list/"

r = requests.get(url)

data = r.json()


free_bike = False
free_park = False
for key, station in data["markers"].items():

    if station["address"] == "TRG MDB":
        #print(station)
        if int(station["station"]["available"]) > 0:
            free_bike = True

    if station["address"] == "STARA CERKEV":
        #print(station)
        if int(station["station"]["free"]) > 0:
            free_park = True

if free_bike and free_park:
    print("Lahko greš z Bicikelj")
else:
    print("Ne moreš se odpeljati")
```

Request Headers

Github Accept header - https://docs.github.com/en/rest/overview/media-types (https://docs.github.com/en/rest/overview/media-types)

Če želimo lahko našemu klicu tudi definiramo headers.

Za primer vzemimo, če na Githubu iščemo vse repositorije, ki vsebujejo besedo requests in so napisani v pythonu.

In [ ]:

```
import requests

response = requests.get('https://api.github.com/search/repositories',
                        params=[('q', 'requests+language:python')],
                       )

data = response.json()
print(data)
print()
print(data["items"][0]["name"])
print(data["items"][0]["description"])
```

Prejšnji search klic lahko doplonemo tako, da specificiramo, da želimo prejeti kje vse v repositoriju smo našli naš iskani vzorec (*requests*). To definiramo s pomočjo headers.

In [ ]:

```
import requests

response = requests.get(
    'https://api.github.com/search/repositories',
    params={'q': 'requests+language:python'},
    headers={'Accept': 'application/vnd.github.v3.text-match+json'},
)

# View the new `text-matches` array which provides information
# about your search term within the results
data = response.json()
print(data)
print()
print(data["items"][0]["name"])
print(data["items"][0]["description"])
print(data["items"][0]["text_matches"])
```

Other HTTP methods

https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods (https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods)

Poleg GET metode so v knjiznjici implementirane na podoben način še POST, PUT, DELETE, HEAD, PATCH, OPTIONS.

- POST - metoda posreduje podatke serverju
- PUT - metoda posodobi določene že obstoječe podatke (večkratno klicanje PUT ima vedno isti rezultat. Večkratno klicanje POST pa ima lahko dodatne posledice)
- DELETE - metoda izbriše podatke

- HEAD - metoda zaprosi naj bodo v odgovoru samo headerji, ki bi bili vrnjeni, če bi ta klic bil GET klic. Primer: Če bi z GET metodo hoteli pridobiti veliko datoteko, lahko prvo uporabimo HEAD in preverimo header **Content-Length**
- PATCH - metoda ima določene inštrukcije, kako posodobiti podatke na serverju
- OPTIONS - metoda pridobi informacijo, katere http metode server podpira

In [ ]:

```
requests.post('https://httpbin.org/post', data={'key':'value'})
requests.put('https://httpbin.org/put', data={'key':'value'})
requests.delete('https://httpbin.org/delete')
requests.head('https://httpbin.org/get')
requests.patch('https://httpbin.org/patch', data={'key':'value'})
requests.options('https://httpbin.org/get')
```

Message Body

Po HTTP specifikacijah, POST, PUT in PATCH metode posredujejo podatke znotraj **message body** namest preko query parametrov.

V requests knjiznjici je to implementirano tako, da naše podatke posredujemo v **data** parameter v obliki dictionary, list of tuples, bytes ali pa file-like objekta.

httpbin.org/post sprejme POST request in nam vrne informacije katere, smo mi posredovali v klicu.

In [ ]:

```
response = requests.post('https://httpbin.org/post', data={'podatek':'HelloWorld!',

print(response.text)
```

Primer, če podatke posredujemo v obliki tuples in a list:

In [ ]:

```
response = requests.post('https://httpbin.org/post', data=[('podatek','HelloWorld!'

print(response.text)
```

Če server od nas zahteva podatke v obliki json jih preprosto podamo metodu kot parameter **json**. Metoda nato nase podatke serializira in doda pravilen Content-Type header.

In [ ]:

```
response = requests.post('https://httpbin.org/post', json={'podatek':'HelloWorld!',

print(response.text)
```

Inspecting our request

Request katerega knjiznjica dejansko naredi lahko pregledamo znotraj našega response objekta.

In [ ]:

```
response = requests.post('https://httpbin.org/post', json={'podatek':'HelloWorld!',

print(response.request)
print("URL:\t", response.request.url)
print("Headers:\t", response.request.headers)
print("Body:\t", response.request.body)
```

Authentication

Zaenkrat smo samo klicali javne API kateri niso potrebovali avtentikacije.

Vendar večina APIjev zahteva neke vrste avtentikacijo, da vedo s kom komunicirajo.

Ponavadi se za avtentikacijo posreduje določene credentials znotraj Authorization headerja oziroma headerja katerega specificira server.

Znotraj knjiznice lahko nase credentials dodamo kot **auth** parameter.

Primer:

Če obiščemo sledeči link https://httpbin.org/basic-auth/user_1/12345678 (https://httpbin.org/basic-auth/user_1/12345678) nas povpraša za username in password, ki ju lahko kot človek vnesemo. (če ne vpraša za vnos username in password odpri link v private oknu)

Če želimo do iste spletne strani dostopati z našo skripto, moramo username in password podati skupaj z našim klicem.

In [ ]:

```
response = requests.get('https://httpbin.org/basic-auth/user_1/12345678', auth=("us
print(response)
print(response.json())
```

In [ ]:

```

```

Sessions

Session's omogočajo, da se določeni podatki ohranijo preko večih različnih klicev.

Vzemimo primer, kjer bomo nastavili random cookie in ne bomo uporabili session-a. Klic na https://httpbin.org/cookies/set (https://httpbin.org/cookies/set) ustvari cookie in nato opravi klic na https://httpbin.org/cookies (https://httpbin.org/cookies) .

In [ ]:

```
import requests

response = requests.get('https://httpbin.org/cookies/set',
                 params={"moj_cookie": "hello world!"})
print(response.text)
```

Če nato sami ponovno opravimo klic na https://httpbin.org/cookies (https://httpbin.org/cookies) vidimo, da našega cooki-a ni več.

In [ ]:

```
response = requests.get('https://httpbin.org/cookies')

print(response.text)
```

Uporabimo sedaj Session objekt.

In [ ]:

```
import requests

session = requests.Session()

response = session.get('https://httpbin.org/cookies/set',
                       params={"moj_cookie": "hello world!"})
print(response.text)
```

Ko opravimo ponoven klic vidimo, da je naš cookie še vedno shranjen.

In [ ]:

```
response = session.get('https://httpbin.org/cookies')
print(response.text)
```

Session objekt nam lahko prav pride, ko želimo ohraniti informacijo o naši avtentikaciji preko večih klicev:

In [ ]:

```
import requests

session = requests.Session()
session.auth = ("user_1", "12345678")

response = session.get('https://httpbin.org/basic-auth/user_1/12345678')
print(response)
print(response.json())
```

Web Scraping with Beautiful Soup

Problem se nam pojavi, če spletne strani nimajo API.

Za primer vzemimo nalogo, kjer želimo pridobiti informacije o episodah za prvi 2 seriji Game of Thrones - No.overall, No. in season, Title, Directed by, Written by, Original air date, U.S. viewers (millions).

> https://en.wikipedia.org/wiki/List_of_Game_of_Thrones_episodes (https://en.wikipedia.org/wiki/List_of_Game_of_Thrones_episodes)

Spletna stran v naši nalogi je napisana v HTML (HyperText Markup Language). Ta zapis spletne strani je posredovan našemu browserju in ta ga spremeni v nam prijazno obliko (dizajn, itd.). Dejanski HTML zapis lahko vidimo s pomočjo "developers tools" - Ctrl+Shift+I (Chrome).

In celotno to kodo (HTML) dobimo, če uporabimo naš zgornji postopek in naredimo GET klic na naš URL.

In [ ]:

```
import requests

url = "https://en.wikipedia.org/wiki/List_of_Game_of_Thrones_episodes"
r = requests.get(url)

print(r.text)
```

In [ ]:

```

```

HTML Quick Overview

uprašat kolk poznajo HTML. Najbrš ne rabm razlagat kako je sestavljen?

In [ ]:

```

```

Dodatna vsebina:

https://www.w3schools.com/html/default.asp (https://www.w3schools.com/html/default.asp)

HTML je sestavljena iz elementov imenovanih **tags**.

Najbolj osnoven tag je `<html> </html>` . Ta tag nam pove, da je vse znotraj njega HTML koda.

Znotraj `<html>` obstajata dva taga:

- `<head></head>` - vsebuje meta podatke o naši spletni strani
- `<body></body>` - vsebuje spletno stran katero vidimo v browserju (naslovi, text, slike, itd.)

```
<html>
    <head>
    </head>

    <body>
    </body>
</html>
```

Tage lahko vstavljamo znotraj drugih tagov, kot sta vstavljena `<head>` in `<body>` znotraj `<html>`. Tagi imajo tako lahko:

- **parent tag** - tag znotraj katerega se nahajajo
- **child tag** - tag, ki se nahaja znotraj njih
- **sibling tag** - tagi, ki se nahajajo v istem parent tag-u

Za dodajanje teksta se najbolj uporablja `<p> Text </p>` tag.

example_01.html

```
<html>
    <head>
    </head>

    <body>
        <p>Webscraping je proces pridobivanja podatkov iz interneta.</p>
    </body>
</html>
```

Če sedaj ponovno odpremo developer's tools lahko točno vidimo naši HTML kodo.

Tag-i imajo tudi določene lastnosti / atribute katere lahko spreminjamo.

Za primer vzemimo tag `<a></a>`, ki deluje kot hiperpovezava / link na drugo spletno stran.

```
<a href="https://www.google.com">Link</a>
```

Tag a ima atribut **href** katerega vrednost je *google.com*, ki nam pove na katero spletno stran naj nas hiperpovezava preusmeri, ko kliknemo na tekst *Link*.

example_02.html

```
<html>
    <head>
    </head>

    <body>
        <p>Webscraping je proces pridobivanja podatkov iz interneta.</p>
        <a href="https://www.google.com">Google brskalnik</a>
    </body>
</html>
```

Dodatno lahko spreminjamo lastnosti tag-ov s pomočjo **class** in **id** atributov. Z njimi lahko spreminjamo izgled naših elementov (barva, velikost, ...) oziroma prikazovanje (element lahko skrijemo, naredimo transparentnega, itd.).

Isti **class** si lahko deli več tag-ov, medtem ko **id** naj bi bil specifičen samo za en tag.

example_03.html

```
<html>
    <head>
        <style>
            #first_text {
                font-size: 20px;
            }

            .red_text {
                color: red;
            }
        </style>
    </head>

    <body>
        <p id="first_text">Webscraping je proces pridobivanja podatkov iz i
nterneta.</p>
        <a href="https://www.google.com">Google brskalnik</a>
        <p class="red_text"> Ta tekst naj bo obarvan rdeče.</p>
    </body>
</html>
```

Poglejmo si sedaj našo nalogo.

S pomočjo developer's tools lahko vidimo, da se podatki za prvo sezono nahajao znotraj `<table>` tag-ov, ki imajo **class="wikitable plainrowheaders wikiepisodetable"**.

```
<table class="wikitable plainrowheaders wikiepisodetable" style="width:10
0%">
```

```
        <tbody ...
    </table>
```

In [ ]:

```
import requests

url = "https://en.wikipedia.org/wiki/List_of_Game_of_Thrones_episodes"
r = requests.get(url)

print(r.text)
```

Sedaj bi lahko sami poiskali vse tabele sezon in ročno našli željene podatke. Vendar je to preveč zakomplicirano.

Za lažje navigiranje po HTML kodi obstaja knjižnjica **BeautifulSoup**.

```
pip install beautifulsoup4
```

In [ ]:

```
import requests
from bs4 import BeautifulSoup

url = "https://en.wikipedia.org/wiki/List_of_Game_of_Thrones_episodes"
r = requests.get(url)

soup = BeautifulSoup(r.text, "html.parser")
print(soup.prettify())
```

Ko ustvarimo BeautifulSoup objekt vanj vnesemo naš tekst in izberemo kateri parser naj uporabi. V našem primeru je to **HTML** ampak obstajajo še drugi, kot je XML.

Nato uporabimo `prettify()` finkcijo za lepši izpis HTML kode.

Za začetek lahko izberemo vse **child tags** naše spletne strani, kar nam bo vrnilo osnovno strukturo `<!DOCTYPE html>` in `<html>` tags.

In [ ]:

```
soup_children = list(soup.children)
print(type(soup_children))
print(len(soup_children))
print(soup_children)
```

Izberimo zadnji element, ki predstavlja našo **html** kodo.

Če preverimo njegov tip vidimo, da je to `bs4.element.Tag` - to je beautiful soup objekt, ki predstavlja naš tag.

In [ ]:

```
html = list(soup.children)[2] # equivalent to soup.html
print(type(html))
print(html)
```

Da vidimo ime našega tag-a lahko uporabimo `tag.name` . Da vidimo njegove atribute lahko uporabimo `tag.attrs`

In [ ]:

```
print(html.name)
print(html.attrs)
```

- class - definira razrede tag-a
- lang - definira jezik v katerem je vsebina tag-a
- dir - specificira smer texta (ltr -> left to right) https://www.w3schools.com/tags/att_dir.asp (https://www.w3schools.com/tags/att_dir.asp)

Da se premaknemo naprej do naše tabele izberemo *children* od našega *html* tag-a. Specifično želimo **body**.

In [ ]:

```
html_children = html.children
for c in html_children:
    print(c.name)
    #print(c)
```

In [ ]:

```
body = list(html.children)[3]
print(body.name)
print(body.attrs)
```

In tako bi lahko nadaljevali dokler ne bi našli naših tabel.

Ampak če želimo najti specifičen tag lahko uporabimo `.find()` metodo. V njej lahko specificiramo ime tag-a katerega iščemo, z `class_` parametrov lahko specificiramo katere **class** vrednosti ima in z `id_` parametrom lahko specificiramo njegov **id** vrednost.

In [ ]:

```
table = body.find("table", class_="wikitable plainrowheaders wikiepisodetable")
print(type(table))
print(table.name)
print(table)
```

Če si pogledamo kako je tabela sestavljena vidimo, da tabela vsebuje 1 child tag **tbody**.

tbody nato vsebuje **tr** tag-e, ki predstavljajo vrstice. tr tag vsebuje **th** oziroma **td** tage, ki predstavljajo stolpce in vsebujejo naše iskane vrednosti.

Izluščimo iz tabele prvi 2 vrstici:

In [ ]:

```
for i in table.children:
    print(i.name)
```

In [ ]:

```
tbody = list(table.children)[0]

for row in list(tbody.children)[:2]:
    print(row.name)
    print(row)
    print()
```

In [ ]:

```
rows = list(tbody.children)[:2]

for row in rows:
    print(row.name)
    for column in row.children:
        print(column.name, column.text)
    print()
```

Da najdemo več kot en tag lahko uporabimo metodo `find_all()` .

In [ ]:

```
tables = soup.find_all("table", class_="wikitable plainrowheaders wikiepisodetable")
print("Našli smo ",len(tables), "tabel.")
print(tables)
```

In [ ]:

```
for table in tables[:2]:
    #print(table)
    rows = table.find_all("tr")
    #print(rows)
    for row in rows[:]:
        #print(row)
        tds = row.find_all("td")
        for td in tds[:]:
            print(td.text)

        print()
    print()
```

Dodatno lahko sedaj pridobimo link vsake episode in odpremo njeno wikipedia spletno stran in poiščemo čas trajanja episode.

Naslov wikipedie strani episode pridobimo v imenu episode, ki je tudi link. Link tag ima atribut **href** katerega vrednost je iskani naslov.

In [ ]:

```
for table in tables[:2]:
    #print(table)
    rows = table.find_all("tr")
    #print(rows)
    for row in rows[1:3]:
        #print(row)
        tds = row.find_all("td")
        for td in tds[:]:
            #print(td.text)
            pass

        title_td = tds[1]
        a = title_td.find("a")
        href = a["href"] #a.attrs["href"]
        print(href)
        url2 = f"https://en.wikipedia.org{href}"
        r2 = requests.get(url2)
        soup2 = BeautifulSoup(r2.text, "html.parser")
        print(soup2.prettify())

        print()
    print()
```

Sedaj ko dostopamo do spletne strani episode tam poiščemo kje se nahaja informacija o dolžini episode.

In [ ]:

```
for table in tables[:2]:
    #print(table)
    rows = table.find_all("tr")
    #print(rows)
    for row in rows[1:3]:
        #print(row)
        tds = row.find_all("td")
        for td in tds[:]:
            print(td.text)

        title_td = tds[1]
        a = title_td.find("a")
        href = a["href"] #a.attrs["href"]
        print(href)
        url2 = f"https://en.wikipedia.org{href}"
        r2 = requests.get(url2)
        soup2 = BeautifulSoup(r2.text, "html.parser")

        infobox_table = soup2.find_all(class_="infobox")[0]
        trs = infobox_table.find_all("tr")
        for tr in trs:
            th = tr.find("th")
            if th and th.text == "Running time":
                td = tr.find("td")
                print(td.text)

        print()
    print()
```

Sedaj, ko imamo podatek o dolžini episode lahko izluščimo točno številko in grafično prikažemo kako se je dolžina episode spreminjala tekom serije.

In [ ]:

```
import re
import matplotlib.pyplot as plt
episode_len = []

for table in tables[:2]:
    #print(table)
    rows = table.find_all("tr")
    #print(rows)
    for row in rows[1:]:
        #print(row)
        tds = row.find_all("td")
        for td in tds[:2]:
            #print(td.text)
            pass

        title_td = tds[1]
        a = title_td.find("a")
        href = a["href"] #a.attrs["href"]
        print(href)
        url2 = f"https://en.wikipedia.org{href}"
        r2 = requests.get(url2)
        soup2 = BeautifulSoup(r2.text, "html.parser")

        infobox_table = soup2.find_all(class_="infobox")[0]
        trs = infobox_table.find_all("tr")
        for tr in trs:
            th = tr.find("th")
            if th and th.text == "Running time":
                td = tr.find("td")
                print(td.text)
                match = re.search("\d+", td.text)
                runtime = match[0]
                print(runtime)
                episode_len.append(int(runtime))

        print()
    print()

plt.plot(range(len(episode_len)) , episode_len)
plt.show()
```

In [ ]:

```

```

In [ ]:

```

```

Naloga:

Ustvarite skripto, ki pridobi informacije o 250 najbolje ocenjenih filmih.

https://www.imdb.com/chart/top/?ref_=nv_mv_250 (https://www.imdb.com/chart/top/?ref_=nv_mv_250)

Skripta naj pridobi naslov filma, oceno filma in trajanje filma. Trajanje filma dobite, če odprete specifični film.

```
Output:
Kaznilnica odrešitve
9.2
2h 22m

Boter
9.1
2h 55m

Boter, II. del
9.0
3h 22m

Vitez teme
9.0
2h 32m

...
```

In [ ]:

```
# Rešitev
import requests
from bs4 import BeautifulSoup
url = "https://www.imdb.com/chart/top/?ref_=nv_mv_250"

r = requests.get(url)
soup = BeautifulSoup(r.content, "html.parser")

table = soup.find_all("tbody", class_="lister-list")[0]

trs = table.find_all("tr")
for tr in trs[:10]:
    title_col = tr.find_all(class_="titleColumn")[0]
    a = title_col.find_all("a")[0]
    title = a.text
    print(title)

    rating_col = tr.find_all(class_="ratingColumn imdbRating")[0]
    rating = rating_col.find_all("strong")[0].text
    print(rating)

    href = a["href"]
    #print(a.attrs["href"])
    url = f"https://www.imdb.com{href}"
    #print(url)

    r2 = requests.get(url)
    soup2 = BeautifulSoup(r2.content, "html.parser")
    #print(soup2.html)

    ul = soup2.find_all("ul", class_="ipc-inline-list")
    #print(len(ul))
    #print(ul)
    lis = ul[0].find_all("li")
    li = lis[-1]
    print(li.text)

    print()
```

In [ ]:

In [ ]:

Web Scraping with Selenium

https://www.browserstack.com/guide/python-selenium-to-run-web-automation-test (https://www.browserstack.com/guide/python-selenium-to-run-web-automation-test)

https://selenium-python.readthedocs.io/ (https://selenium-python.readthedocs.io/)

1. https://www.geeksforgeeks.org/selenium-python-tutorial/ (https://www.geeksforgeeks.org/selenium-python-tutorial/)

Selenium je orodje, s katerim lahko naš program kontrolira browser (Chrome, Mozzila, ...). Selenium je napisan v večih jezikih (Java, C#, ...) med drugim tudi v Pythonu.

Uporablja se za pisanje avtomatičnih testov za vaše aplikacije oziroma, če je potrebno pridobiti podatke iz bolj zaščitenih spletnih strani oziroma spletnih strani, ki uporabljajo veliko JavaScript-a.

```
pip install selenium
```

Za delovanje potrebujemo še browser driver:

> https://selenium-python.readthedocs.io/installation.html (https://selenium-python.readthedocs.io/installation.html)

1.5. Drivers

> Selenium requires a driver to interface with the chosen browser. Firefox, for example, requires geckodriver, which needs to be installed before the below examples can be run. Make sure it's in your PATH, e. g., place it in /usr/bin or /usr/local/bin.

> Failure to observe this step will give you an error selenium.common.exceptions.WebDriverException: Message: 'geckodriver' executable needs to be in PATH.

> Other supported browsers will have their own drivers available. Links to some of the more popular browser drivers follow.

- Chrome: https://sites.google.com/chromium.org/driver/ (https://sites.google.com/chromium.org/driver/)
  - To find chrome version go to -> RIght upper corner, "About chrome"
- Edge: https://developer.microsoft.com/en-us/microsoft-edge/tools/webdriver/ (https://developer.microsoft.com/en-us/microsoft-edge/tools/webdriver/)
- Firefox: https://github.com/mozilla/geckodriver/releases (https://github.com/mozilla/geckodriver/releases)
- Safari: https://webkit.org/blog/6900/webdriver-support-in-safari-10/ (https://webkit.org/blog/6900/webdriver-support-in-safari-10/)

For more information about driver installation, please refer the official documentation.

(Dela na Google Chrome). Če vrže napako, da maš samo **data;** v address bar-u pol lahko inštaliramo starejšo verzijo chrome:

```
$ wget -O /tmp/chrome.deb https://dl.google.com/linux/chrome/deb/pool/main/g/google-chrome-stable/google-chrome-stable_96.0.4664.45-1_amd64.deb
```

> 1. Zdownloada starejšo verzijo google chrome v /temp/chrome.deb

```
$ sudo apt install -y /tmp/chrome.deb
```

> 2. Inštalira chrome

```
$ rm /tmp/chrome.deb
```

> 3. Zbriše zdownloadano datoteko

Uporabmo realni primer in sproti kažemo še druge možnosti (druge možnosti iskanja elementov, waits, itd...)

In [ ]:

```

```

Tekom naše naloge bomo parsali podatke iz sledeče spletne strani - https://livetoken.co/listings/topshot (https://livetoken.co/listings/topshot)

Na tej spletni strani si lahko pogledamo market z NBA Top Shot Moments - na splošno povedano so izseki iz NBA tekem, katere lahko zbiralci kupujejo in prodajajo.

Naš cilj je ustvariti skripto, ki preveri cene naših momentov.

```
my_portfolio = [
        {"Name": "LUKA DONČIĆ" ,
         "Type": "Assist",
         "Date": "1/17/2021",},

        {"Name": "JAMYCHAL GREEN",
         "Type": "Dunk",
         "Date": "1/3/2021",},

        {"Name": "T.J. MCCONNELL",
         "Type": "Assist",
         "Date": "12/23/2020",},
        ]
```

Da dobimo ceno moramo:

1. Obiskati stran - https://livetoken.co/listings/topshot (https://livetoken.co/listings/topshot)
2. V prvi drop down vpisati ime igralca
3. V drugi dropdown vpisati datum in nato izbrati naš moment (ponavadi je en moment na datum)
4. Sortirati od najcenejšega do najdražjega (v kolikor to ni default)
5. Pridobiti ceno

In [ ]:

```
```

1. Obiskati spletno stran

Začeli bomo s tem, da zaženemo naš browser in odpremo spletno stran.

In [ ]:

```
import time

from selenium import webdriver

my_portfolio = [
        {"Name": "LUKA DONČIĆ" ,
         "Type": "Assist",
         "Date": "1/17/2021",},

        {"Name": "JAMYCHAL GREEN",
         "Type": "Dunk",
         "Date": "1/3/2021",},

        {"Name": "T.J. MCCONNELL",
         "Type": "Assist",
         "Date": "12/23/2020",},
]

driver = webdriver.Chrome("./chromedriver_linux_96-0-4664-45")
driver.get("https://livetoken.co/listings/topshot")
time.sleep(5)
driver.close()
```

S pomočjo `webdriver.Chrome()` metode definiramo kater browser bomo uporabljali in kje se nahaja naš driver.

Nato s `driver.get()` izvedemo klic na določeno spletno stran. WeDriver bo počakal dokler strani ni naložen (dokler se ni sprožil **onload** event) in nato vrnil kontrolo naši python skripti. Tukaj je potrebno paziti, če stran uporablja veliko AJAX-a, ker naš webdriver ne bo znal določiti kdaj je stran dejansko popolnoma naložena.

> **onload event** se sproži, ko se naložijo vse dependent zadeve, kot so slike, css, itd...
> https://developer.mozilla.org/en-US/docs/Web/API/Window/load_event
> (https://developer.mozilla.org/en-US/docs/Web/API/Window/load_event)

Nato počakamo 5 sekund in ugasnemo naš driver.

Vidimo tudi, da smo dobili `DeprecationWarning` .

```
DeprecationWarning: executable_path has been deprecated, please pass
in a Service object
```

Z novejšo različico seleniuma je potek zagona driverja drugačen.

In [ ]:

```
import time
from selenium import webdriver
from selenium.webdriver.chrome.service import Service

my_portfolio = [
        {"Name": "LUKA DONČIĆ" ,
         "Type": "Assist",
         "Date": "1/17/2021",},

        {"Name": "JAMYCHAL GREEN",
         "Type": "Dunk",
         "Date": "1/3/2021",},

        {"Name": "T.J. MCCONNELL",
         "Type": "Assist",
         "Date": "12/23/2020",},
]

s = Service("./chromedriver_linux_96-0-4664-45")
with webdriver.Chrome(service=s) as driver:
    driver.get("https://livetoken.co/listings/topshot")
    time.sleep(5)
```

Sedaj prvo ustvarimo `Service()` objekt v katerem definiramo pot do našega driverja in nato ta objekt pošljemo `webdriver.Chrome()` .

Dodatno smo naš driver odprli z `with` stavkom, tako da se po koncu naše skripte avtomatično ugasne.

Za lepšo preglednost lahko naš browser odpremo v *maximised* načinu - čez cel ekran.

In [ ]:

```
import time
from selenium import webdriver
from selenium.webdriver.chrome.service import Service

my_portfolio = [
        {"Name": "LUKA DONČIĆ" ,
         "Type": "Assist",
         "Date": "1/17/2021",},

        {"Name": "JAMYCHAL GREEN",
         "Type": "Dunk",
         "Date": "1/3/2021",},

        {"Name": "T.J. MCCONNELL",
         "Type": "Assist",
         "Date": "12/23/2020",},
]

s = Service("./chromedriver_linux_96-0-4664-45")
with webdriver.Chrome(service=s) as driver:
    driver.maximize_window()
    driver.get("https://livetoken.co/listings/topshot")
    time.sleep(5)
```

In [ ]:

```
```

Vpisati ime v prvi dropdown

Naslednji korak je izbrati prvi dropdown meni in vanj vpisati ime igralca.

V seleniumu lahko do elementov dostopamo na različne načine. Nazaj bomo vedno dobili `WebElement` oziroma list `WebElement-ov` .

Do njih dostopamo preko metode `find_element(by=BY, value=value)` , ki nam vrne prvi element, ki pade v naš kriterij. Oziroma preko metode `find_elements()` , ki nam vrne list elementov, ki padejo v naš kriterij.

Naše kriterije iskanja določamo s pomočjo parametra **by** in **value**. Iščemo lahko preko **BY.ID**, **BY.CLASS_NAME**, **BY.CSS_SELECTOR**, **BY.TAG_NAME**, itd.

V kolikor ne najdemo nobenega elementa dobimo `NoSuchElementException` .

> CSS SELECTORS: https://www.w3schools.com/cssref/css_selectors.asp (https://www.w3schools.com/cssref/css_selectors.asp)

Vse te vrednosti lahko najdemo s pomočjo **developers tools** - Chrome (F12 oziroma desni klik in *inspect*).

Dodatno bomo še izpisali HTML kodo našega prejetega elementa.

In [ ]:

```
import time
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
# vvvvv     HERE     vvvvv
from selenium.webdriver.common.by import By
# ^^^^^     HERE     ^^^^^

my_portfolio = [
        {"Name": "LUKA DONČIĆ" ,
         "Type": "Assist",
         "Date": "1/17/2021",},

        {"Name": "JAMYCHAL GREEN",
         "Type": "Dunk",
         "Date": "1/3/2021",},

        {"Name": "T.J. MCCONNELL",
         "Type": "Assist",
         "Date": "12/23/2020",},
]

s = Service("./chromedriver_linux_96-0-4664-45")
with webdriver.Chrome(service=s) as driver:
    driver.maximize_window()
    driver.get("https://livetoken.co/listings/topshot")

    # vvvvv     HERE     vvvvv
    el = driver.find_element(by=By.ID, value="title")
    el = driver.find_element(by=By.CLASS_NAME, value="navLinks")
    el = driver.find_element(by=By.TAG_NAME, value="li")
    el = driver.find_element(by=By.CSS_SELECTOR, value=".navLinks:last-child")

    print(el.get_attribute('outerHTML'))
    # ^^^^^     HERE     ^^^^^

    time.sleep(5)
```

Da dobimo več element naenkrat lahko rečemo:

In [ ]:

```
import time
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By

my_portfolio = [
        {"Name": "LUKA DONČIĆ" ,
         "Type": "Assist",
         "Date": "1/17/2021",},

        {"Name": "JAMYCHAL GREEN",
         "Type": "Dunk",
         "Date": "1/3/2021",},

        {"Name": "T.J. MCCONNELL",
         "Type": "Assist",
         "Date": "12/23/2020",},
]

s = Service("./chromedriver_linux_96-0-4664-45")
with webdriver.Chrome(service=s) as driver:
    driver.maximize_window()
    driver.get("https://livetoken.co/listings/topshot")

    # vvvvv     HERE     vvvvv
    elements = driver.find_elements(by=By.CLASS_NAME, value="navLinks")
    for el in elements:
        print(el.get_attribute('outerHTML'))
    # ^^^^^     HERE     ^^^^^

    time.sleep(5)
```

Če pogledamo HTML kodo našega dropdown menija je sledeča:

```
    <div class="vs__selected-options">
        <span class="vs__selected">
                All Players
        <!----></span>
        <input aria-autocomplete="list" aria-labelledby="vs1__combobox" aria-co
    ntrols="vs1__listbox" type="search" autocomplete="off" class="vs__search">
    </div>
```

In [ ]:

```
import time
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By

my_portfolio = [
        {"Name": "LUKA DONČIĆ" ,
         "Type": "Assist",
         "Date": "1/17/2021",},

        {"Name": "JAMYCHAL GREEN",
         "Type": "Dunk",
         "Date": "1/3/2021",},

        {"Name": "T.J. MCCONNELL",
         "Type": "Assist",
         "Date": "12/23/2020",},
]

s = Service("./chromedriver_linux_96-0-4664-45")
with webdriver.Chrome(service=s) as driver:
    driver.maximize_window()
    driver.get("https://livetoken.co/listings/topshot")

    # vvvvv     HERE     vvvvv
    name_field = driver.find_element(by=By.CSS_SELECTOR, value="div.vs__selected-op
    print(name_field.get_attribute('outerHTML'))
    # ^^^^^     HERE     ^^^^^

    time.sleep(5)
```

Sedaj, ko lahko dostopamo do našega dropdown menija, hočemo vanj vpisati ime košarkaša.

Da simuliramo vnos črk v nek element uporabimo metodo `send_keys()` .

In [ ]:

```
import time
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By

my_portfolio = [
        {"Name": "LUKA DONČIĆ" ,
         "Type": "Assist",
         "Date": "1/17/2021",},

        {"Name": "JAMYCHAL GREEN",
         "Type": "Dunk",
         "Date": "1/3/2021",},

        {"Name": "T.J. MCCONNELL",
         "Type": "Assist",
         "Date": "12/23/2020",},
]

s = Service("./chromedriver_linux_96-0-4664-45")
with webdriver.Chrome(service=s) as driver:
    driver.maximize_window()
    driver.get("https://livetoken.co/listings/topshot")

    name_field = driver.find_element(by=By.CSS_SELECTOR, value="div.vs__selected-op
    print(name_field.get_attribute('outerHTML'))
    # vvvvv     HERE     vvvvv
    name_field.send_keys(my_portfolio[0]["Name"])
    # ^^^^^     HERE     ^^^^^

    time.sleep(5)
```

Sedaj dobimo `ElementNotInteractableException` . To pomeni, da v element še ne moremo vnašati črk. Ponavadi kakšen drug element stoji v ospredju (na primer gumb, ki čaka da sprejmemo ali zavrnemo piškotke) in moramo počakat, da izgine oziroma ga sami odklikati stran.

Najbolj osnovna rešitev je, da preprosto počakamo še nekaj časa:

In [ ]:

```
import time
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By

my_portfolio = [
        {"Name": "LUKA DONČIĆ" ,
         "Type": "Assist",
         "Date": "1/17/2021",},

        {"Name": "JAMYCHAL GREEN",
         "Type": "Dunk",
         "Date": "1/3/2021",},

        {"Name": "T.J. MCCONNELL",
         "Type": "Assist",
         "Date": "12/23/2020",},
]

s = Service("./chromedriver101_0_4951_41")
with webdriver.Chrome(service=s) as driver:
    driver.maximize_window()
    driver.get("https://livetoken.co/listings/topshot")

    name_field = driver.find_element(by=By.CSS_SELECTOR, value="div.vs__selected-op
    print(name_field.get_attribute('outerHTML'))
    # vvvvv     HERE     vvvvv
    time.sleep(10)
    name_field.send_keys(my_portfolio[0]["Name"])
    # ^^^^^     HERE     ^^^^^

    time.sleep(5)
```

In [ ]:

```

```

Namesto čakanja lahko v seleniumu določimo **Explicit Wait** oziroma specifični vzrok čakanja.

V našem primeru čakamo, da naš element postane "clickable".

In [ ]:

```
import time
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By
# vvvvv     HERE     vvvvv
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
# ^^^^^     HERE     ^^^^^

my_portfolio = [
        {"Name": "LUKA DONČIĆ" ,
         "Type": "Assist",
         "Date": "1/17/2021",},

        {"Name": "JAMYCHAL GREEN",
         "Type": "Dunk",
         "Date": "1/3/2021",},

        {"Name": "T.J. MCCONNELL",
         "Type": "Assist",
         "Date": "12/23/2020",},
]

s = Service("./chromedriver_linux_96-0-4664-45")
with webdriver.Chrome(service=s) as driver:
    driver.maximize_window()
    driver.get("https://livetoken.co/listings/topshot")

    name_field = driver.find_element(by=By.CSS_SELECTOR, value="div.vs__selected-op
    print(name_field.get_attribute('outerHTML'))
    # vvvvv     HERE     vvvvv
    WebDriverWait(driver, 10).until(EC.element_to_be_clickable(name_field))
    name_field.send_keys(my_portfolio[0]["Name"])
    # ^^^^^     HERE     ^^^^^

    time.sleep(5)
```

Naš driver bo sedaj eksplicitno počakal 10sekund, da je naš element "clickable". Če naš element ne postane clickable, driver vrže error.

Sedaj moramo klikniti "ENTER" in nato ponoviti postopek še za **All Moments** element.

Privzamemo, da za specifičen datum obstaja le ena vrednost tako, da bomo vpisali le datum.

In [ ]:

```
import time
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
# vvvvv     HERE     vvvvv
from selenium.webdriver.common.keys import Keys
# ^^^^^     HERE     ^^^^^

my_portfolio = [
        {"Name": "LUKA DONČIĆ" ,
         "Type": "Assist",
         "Date": "1/17/2021",},

        {"Name": "JAMYCHAL GREEN",
         "Type": "Dunk",
         "Date": "1/3/2021",},

        {"Name": "T.J. MCCONNELL",
         "Type": "Assist",
         "Date": "12/23/2020",},
]

s = Service("./chromedriver_linux_96-0-4664-45")
with webdriver.Chrome(service=s) as driver:
    driver.maximize_window()
    driver.get("https://livetoken.co/listings/topshot")

    name_field = driver.find_element(by=By.CSS_SELECTOR, value="div.vs__selected-op
    print(name_field.get_attribute('outerHTML'))
    WebDriverWait(driver, 10).until(EC.element_to_be_clickable(name_field))
    name_field.send_keys(my_portfolio[0]["Name"])
    # vvvvv     HERE     vvvvv
    name_field.send_keys(Keys.ENTER)
    # ^^^^^     HERE     ^^^^^

    time.sleep(5)
```

Vpis datuma v drugi dropdown meni

In pa ponovimo postopek še za drugi dropdown meni. Prvo moramo vnesti ime, nato se nam pokaže drugi dropdown meni.

HTML koda izgleda nekako sledeče:

```
<div class="vs__selected-options"><span class="vs__selected">
            All Moments
           <!----></span> <input aria-autocomplete="list" aria-labelledby=
"vs3__combobox" aria-controls="vs3__listbox" type="search" autocomplete="of
f" class="vs__search"></div>
```

In [ ]:

```
import time
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.common.keys import Keys

my_portfolio = [
        {"Name": "LUKA DONČIĆ" ,
         "Type": "Assist",
         "Date": "1/17/2021",},

        {"Name": "JAMYCHAL GREEN",
         "Type": "Dunk",
         "Date": "1/3/2021",},

        {"Name": "T.J. MCCONNELL",
         "Type": "Assist",
         "Date": "12/23/2020",},
]

s = Service("./chromedriver_linux_96-0-4664-45")
with webdriver.Chrome(service=s) as driver:
    driver.maximize_window()
    driver.get("https://livetoken.co/listings/topshot")

    name_field = driver.find_element(by=By.CSS_SELECTOR, value="div.vs__selected-op
    print(name_field.get_attribute('outerHTML'))
    WebDriverWait(driver, 10).until(EC.element_to_be_clickable(name_field))
    name_field.send_keys(my_portfolio[0]["Name"])
    name_field.send_keys(Keys.ENTER)

    # vvvvv     HERE     vvvvv
    all_moments_field = driver.find_elements(by=By.CSS_SELECTOR, value="div.vs__sel
    print(all_moments_field.get_attribute('outerHTML'))
    WebDriverWait(driver, 10).until(EC.element_to_be_clickable(all_moments_field))
    all_moments_field.send_keys(my_portfolio[0]["Date"])
    time.sleep(2) # Ker če ne se prehitro pritisne ENTER, medtem ko v ozadju dropdo
    all_moments_field.send_keys(Keys.ENTER)
    # ^^^^^     HERE     ^^^^^

    time.sleep(5)
```

Pridobitev cene

Sedaj moramo pridobiti informacijo o prvem in drugem najcenejšem momentu in izračunati našo ceno (ki je povprečje teh dveh).

Vse cene so v HTML:

```
    <div data-v-2f19ebab="" class="cost regularAsk">$7</div>
```

Najcenejši moment ima namesto **regularAsk** class **lowestAsk**.

In [ ]:

```
import time
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.common.keys import Keys

my_portfolio = [
        {"Name": "LUKA DONČIĆ" ,
         "Type": "Assist",
         "Date": "1/17/2021",},

        {"Name": "JAMYCHAL GREEN",
         "Type": "Dunk",
         "Date": "1/3/2021",},

        {"Name": "T.J. MCCONNELL",
         "Type": "Assist",
         "Date": "12/23/2020",},
]

s = Service("./chromedriver_linux_96-0-4664-45")
with webdriver.Chrome(service=s) as driver:
    driver.maximize_window()
    driver.get("https://livetoken.co/listings/topshot")

    name_field = driver.find_element(by=By.CSS_SELECTOR, value="div.vs__selected-op
    print(name_field.get_attribute('outerHTML'))
    WebDriverWait(driver, 10).until(EC.element_to_be_clickable(name_field))
    name_field.send_keys(my_portfolio[0]["Name"])
    name_field.send_keys(Keys.ENTER)

    all_moments_field = driver.find_elements(by=By.CSS_SELECTOR, value="div.vs__sel
    print(all_moments_field.get_attribute('outerHTML'))
    WebDriverWait(driver, 10).until(EC.element_to_be_clickable(all_moments_field))
    all_moments_field.send_keys(my_portfolio[0]["Date"])
    time.sleep(2) # Ker če ne se prehitro pritisne ENTER, medtem ko v ozadju dropdo
    all_moments_field.send_keys(Keys.ENTER)

    # vvvvv     HERE     vvvvv
    time.sleep(5)
    prices = driver.find_elements(by=By.CSS_SELECTOR, value="div.cost")
    price_1 = prices[0].get_attribute("innerText")
    price_2 = prices[1].get_attribute("innerText")
    print(price_1)
    print(price_2)
    # ^^^^^     HERE     ^^^^^

    time.sleep(5)
```

Text vrednosti imamo, sedaj je potrebno odstraniti $ znak in zadeve pretvoriti v dejanske številske vrednosti in nato izračunati našo prodajno ceno, ki je povprečna cena teh dveh.

In [ ]:

```
import time
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.common.keys import Keys

my_portfolio = [
        {"Name": "LUKA DONČIĆ" ,
         "Type": "Assist",
         "Date": "1/17/2021",},

        {"Name": "JAMYCHAL GREEN",
         "Type": "Dunk",
         "Date": "1/3/2021",},

        {"Name": "T.J. MCCONNELL",
         "Type": "Assist",
         "Date": "12/23/2020",},
]

s = Service("./chromedriver_linux_96-0-4664-45")
with webdriver.Chrome(service=s) as driver:
    driver.maximize_window()
    driver.get("https://livetoken.co/listings/topshot")

    name_field = driver.find_element(by=By.CSS_SELECTOR, value="div.vs__selected-op
    print(name_field.get_attribute('outerHTML'))
    WebDriverWait(driver, 10).until(EC.element_to_be_clickable(name_field))
    name_field.send_keys(my_portfolio[0]["Name"])
    name_field.send_keys(Keys.ENTER)

    all_moments_field = driver.find_elements(by=By.CSS_SELECTOR, value="div.vs__sel
    print(all_moments_field.get_attribute('outerHTML'))
    WebDriverWait(driver, 10).until(EC.element_to_be_clickable(all_moments_field))
    all_moments_field.send_keys(my_portfolio[0]["Date"])
    time.sleep(2) # Ker če ne se prehitro pritisne ENTER, medtem ko v ozadju dropdo
    all_moments_field.send_keys(Keys.ENTER)

    time.sleep(5)
    prices = driver.find_elements(by=By.CSS_SELECTOR, value="div.cost")
    price_1 = prices[0].get_attribute("innerText")
    price_2 = prices[1].get_attribute("innerText")
    print(price_1)
    print(price_2)

    # vvvvv     HERE     vvvvv
    price_1 = int(price_1.strip("$"))
    price_2 = int(price_2.strip("$"))
    my_price = (price_1 + price_2) / 2
    print(f"My price for {my_portfolio[0]['Name']} {my_portfolio[0]['Date']}: {my_p
    # ^^^^^     HERE     ^^^^^

    time.sleep(5)
```

Dodamo zadevo v for loop, ki najde cene za naš celoten portfolio.

In [ ]:

```
import time
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.common.keys import Keys

my_portfolio = [
        {"Name": "LUKA DONČIĆ" ,
         "Type": "Assist",
         "Date": "1/17/2021",},

        {"Name": "JAMYCHAL GREEN",
         "Type": "Dunk",
         "Date": "1/3/2021",},

        {"Name": "T.J. MCCONNELL",
         "Type": "Assist",
         "Date": "12/23/2020",},
]

s = Service("./chromedriver_linux_96-0-4664-45")
with webdriver.Chrome(service=s) as driver:
    driver.maximize_window()
    driver.get("https://livetoken.co/listings/topshot")

    # vvvvv     HERE     vvvvv
    for moment in my_portfolio:
    # ^^^^^     HERE     ^^^^^
        name_field = driver.find_element(by=By.CSS_SELECTOR, value="div.vs__selecte
        print(name_field.get_attribute('outerHTML'))
        WebDriverWait(driver, 10).until(EC.element_to_be_clickable(name_field))
        # vvvvv     HERE     vvvvv
        name_field.send_keys(moment["Name"])
        # ^^^^^     HERE     ^^^^^
        name_field.send_keys(Keys.ENTER)

        all_moments_field = driver.find_elements(by=By.CSS_SELECTOR, value="div.vs_
        print(all_moments_field.get_attribute('outerHTML'))
        WebDriverWait(driver, 10).until(EC.element_to_be_clickable(all_moments_fiel
        # vvvvv     HERE     vvvvv
        all_moments_field.send_keys(moment["Date"])
        # ^^^^^     HERE     ^^^^^
        time.sleep(5) # Ker če ne se prehitro pritisne ENTER, medtem ko v ozadju dr
        all_moments_field.send_keys(Keys.ENTER)

        time.sleep(5)
        prices = driver.find_elements(by=By.CSS_SELECTOR, value="div.cost")
        price_1 = prices[0].get_attribute("innerText")
        price_2 = prices[1].get_attribute("innerText")
        print(price_1)
        print(price_2)

        price_1 = int(price_1.strip("$"))
        price_2 = int(price_2.strip("$"))
        my_price = (price_1 + price_2) / 2
        # vvvvv     HERE     vvvvv
```

```
        print(f"My price for {moment['Name']} {moment['Date']}: {my_price}")
        # ^^^^^     HERE     ^^^^^


    time.sleep(5)
```

Za konec bomo zadevo zagnali še v **headless** načinu, kar pomeni, da se ne no odprlo nobeno okno in bo program deloval "v ozadju".

In [ ]:

```
import time
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.common.keys import Keys
# vvvvv     HERE     vvvvv
from selenium.webdriver.chrome.options import Options
# ^^^^^     HERE     ^^^^^

my_portfolio = [
        {"Name": "LUKA DONČIĆ" ,
         "Type": "Assist",
         "Date": "1/17/2021",},

        {"Name": "JAMYCHAL GREEN",
         "Type": "Dunk",
         "Date": "1/3/2021",},

        {"Name": "T.J. MCCONNELL",
         "Type": "Assist",
         "Date": "12/23/2020",},
]

s = Service("./chromedriver_linux_96-0-4664-45")
# vvvvv     HERE     vvvvv
chrome_options = Options()
chrome_options.add_argument("--headless")
with webdriver.Chrome(service=s, options=chrome_options) as driver:
# ^^^^^     HERE     ^^^^^
    driver.maximize_window()
    driver.get("https://livetoken.co/listings/topshot")

    for moment in my_portfolio:
        name_field = driver.find_element(by=By.CSS_SELECTOR, value="div.vs__selecte
        print(name_field.get_attribute('outerHTML'))
        WebDriverWait(driver, 10).until(EC.element_to_be_clickable(name_field))
        name_field.send_keys(moment["Name"])
        name_field.send_keys(Keys.ENTER)

        all_moments_field = driver.find_elements(by=By.CSS_SELECTOR, value="div.vs_
        print(all_moments_field.get_attribute('outerHTML'))
        WebDriverWait(driver, 10).until(EC.element_to_be_clickable(all_moments_fiel
        all_moments_field.send_keys(moment["Date"])
        time.sleep(5) # Ker če ne se prehitro pritisne ENTER, medtem ko v ozadju dr
        all_moments_field.send_keys(Keys.ENTER)

        time.sleep(5)
        prices = driver.find_elements(by=By.CSS_SELECTOR, value="div.cost")
        price_1 = prices[0].get_attribute("innerText")
        price_2 = prices[1].get_attribute("innerText")
        print(price_1)
        print(price_2)

        price_1 = int(price_1.strip("$"))
        price_2 = int(price_2.strip("$"))
        my_price = (price_1 + price_2) / 2
        print(f"My price for {moment['Name']} {moment['Date']}: {my_price}")
```

```
time.sleep(5)
```

"Headless" rešitve za Firefox so malo težje:

> https://stackoverflow.com/questions/5370762/how-to-hide-firefox-window-selenium-webdriver (https://stackoverflow.com/questions/5370762/how-to-hide-firefox-window-selenium-webdriver)

In [ ]:

```

```