Școala
informală
de IT

Python Development

Web scraping

Cuprins

Ce este web scraping?

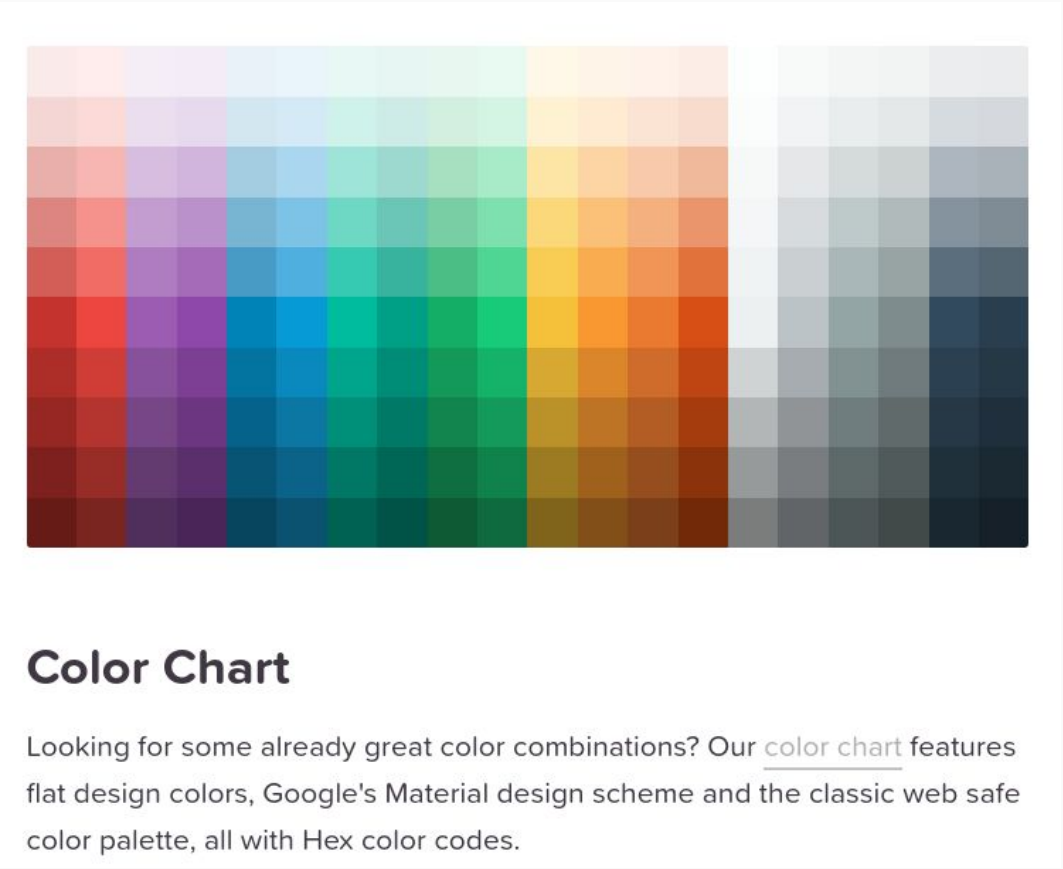
Web scraping

Ce este web scraping?

- Web scraping-ul este procesul de a aduna informații de pe internet. Chiar și copiatul versurilor unei melodii favorite poate fi considerat web scraping.
- Totuși, web scraping-ul presupune un proces automat de a obține informații.
- Există site-uri care nu au probleme cu aceste tool-uri care adună informații în mod automat, dar unele nu sunt de acord cu această tehnică.
- **Atenție!** Înainte de a vă apuca de un proiect de web scraping asigurați-vă că nu încălcați **Terms of Services** aplicațiilor respective. Mai multe detalii găsiți în link-ul următor:
    - https://benbernardblog.com/web-scraping-and-crawling-are-perfectly-legal-right/
- Unele aplicații oferă un API (**A**pplication **P**rogramming **I**nterfaces) care expune datele într-o manieră predefinită. Avantajul consumării acestor API-uri oferă avantajul de a lucra direct cu un tip de date ca JSON sau XML, evitând astfel parsarea HTML-ului.

Ce este web scraping?

- Rolul acestei prezentări este să obțină datele despre culori de pe site-ul https://htmlcolorcodes.com/. Fiecare culoare are un cod hexazecimal, valori RGB (Red, Green, Blue) și valori HSL (Hue, Saturation, Lightness).



- De la această adresă vom extrage informațiile despre culorile din chart-ul prezentat în imaginea alăturată.
- Informațiile care ne interesează pentru prima culoare le vedeți în JSON-ul de mai jos:


```
{
    "hex": "#F9EBEA",
    "rgb": {
        "r": 249,
        "g": 235,
        "b": 234
    },
    "hsl": {
        "h": 6,
        "s": "54%",
        "l": "95%"
    }
},
```

Tool-uri

Web scraping

Tool-uri

- În zilele noastre există două tipuri de site-uri:
  - pagini statice - acestea sunt randate de către server, iar request-ul către un server va avea ca răspuns un document HTML. Ulterior acesta va fi randat în browser.
  - aplicații web - acestea sunt randate de către browser. Request-ul către un server întoarce fie un document HTML minimal cu conținut JavaScript fie direct un fișier JavaScript. Ulterior acesta va fi rulat de către browser, iar conținutul paginii va fi construit dinamic.
- Pentru că există din ce în ce mai puține pagini statice, al căror document HTML este randat direct în browser, vom folosi **selenium** pentru navigarea în browser.
- Pentru a accesa conținutul din pagină vom folosi **BeautifulSoup**.

Tool-uri

Selenium

- Pentru a folosi selenium avem nevoie să instalăm următoarele:
  - un browser la alegere
  - un webdriver pentru browser-ul respectiv (versiunea webdriver-ului și a browser-ului trebuie să fie aceeași).
  - modulul selenium pentru Python (**pip install selenium**)
- Mai multe detalii găsiți la adresa https://selenium-python.readthedocs.io/installation.html, pașii 1.2 & 1.3

BeautifulSoup

- Este o librărie de Python ce are rol de a extrage date din documente HTML și XML.
- Pentru instalare: **pip install beautifulsoup4**

Regular expressions

Web scraping

Regular expressions

- Pentru a extrage informația din atributele discutate anterior va trebui să folosim regula expressions (regex).
- Expresiile regulate sunt un mecanism care permit căutarea unui pattern într-un anumit string.
- Pentru a folosi regular expressions în Python avem nevoie de modulul **re** (***import re***).
- Există două modalități de a identifica secvențele dorite într-un șir de caractere:
  - folosind metoda search (**re.search**) - ajută la identificarea primului match al pattern-ului
  - folosind metoda findall (**re.findall**) - ajută la identificare multiplelor match-uri ale pattern-ului. Această metodă o vom folosi pentru extragerea numerelor din atributele elementelor HTML.
- Mai multe detalii puteți găsi în documentația oficială:

  https://docs.python.org/3/library/re.html

- Un tool online pentru verificarea pattern-urilor puteți găsi la următoarea adresă:

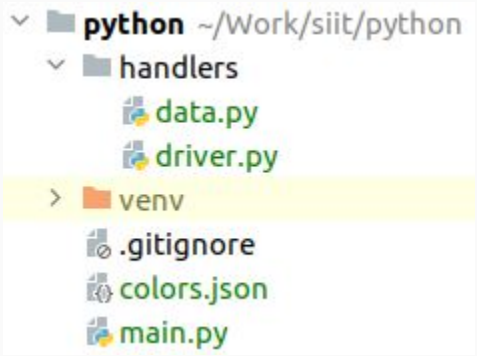
  https://regex101.com/

Codebase

Web scraping

Codebase

- Structura de fișiere va fi următoarea:



- **main.py** - acesta va fi entrypoint-ul programului nostru
- **colors.json** - acesta va fi output-ul programului nostru
- **handlers/data.py** - va conține funcțiile necesare prelucrării datelor din elementele HTML
- **handlers/driver.py** - va conține funcțiile necesare navigării în aplicația web.

Codebase

```python
from selenium import webdriver
from bs4 import BeautifulSoup
from handlers.data import write_output, parse_colors
from handlers.driver import accept_cookies

source = 'https://htmlcolorcodes.com/'


if __name__ == '__main__':
    # Start a new Selenium driver with Firefox.
    driver = webdriver.Firefox()

    # Get content of desired website.
    driver.get(source)

    # Accept the cookies when prompted.
    accept_cookies(driver)

    # Set page source to BeautifulSoup so we can parse the HTML data.
    soup = BeautifulSoup(driver.page_source, features='html.parser')
    colors = parse_colors(soup)

    # Write output to file
    write_output(source, colors)

    # Close the driver
    driver.close()
```

- Codul din fișierul **main.py** are următoarea structură:
  - creează un webdriver de Firefox
  - obține aplicația de la adresa indicată de variabila ***source***.
  - apelează metoda care va accepta user consent-ul
  - creează un obiect **BeautifulSoup** pe baza conținutului documentului HTML obținut pe baza driver-ului de Selenium.
  - parsează elementele HTML pentru obținerea informațiilor despre culori.
  - apelează metoda ce va scrie informațiile obținute în fișierul ***colors.json***.
  - închide driver-ul.

Codebase

```python
from selenium.webdriver.support.wait import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.common.by import By


def accept_cookies(driver):
    """
    Agree with Cookies terms and conditions.
    :param driver: The Selenium driver used to browse the application
    :return: None
    """
    try:
        agree_button = WebDriverWait(driver, 10).until(
            EC.element_to_be_clickable((By.XPATH, './/button[text()="AGREE"]')))
    except BaseException as e:
        # You can do something with the exception.
        agree_button = None

    if agree_button:
        agree_button.click()
```

- Metoda **accept_cookies** are rolul să aștepte timp de 10 secunde ca butonul ce conține textul **AGREE** să apară în pagină.
- Dacă acest buton apare este preluat în variabila **agree_button**.
- În final, dacă acesta nu are o valoare nulă, se va face click pe el.
- Metoda folosită pentru selectarea elementului este **XPath**.
- Pentru așteptare se folosește WebDriverWait care primește doi parametrii:
  - driver-ul de Selenium
  - numărul de secunde pe care să-l aștepte

Codebase

```python
rgb_keys = ('r', 'g', 'b')
hsl_keys = ('h', 's', 'l')


def parse_colors(soup):
    colors = []

    # Get the chart element by ID
    chart_element = soup.find('header', {'id': 'flat'})

    # Parse all div elements with class 'color-group'
    for color_group in chart_element.findAll('div', class_='color-group'):
        # Parse all div elements with class 'js-color'
        for color_element in color_group.findAll('div', class_='js-color'):
            # Append element with data from HTML elements to `colors`.
            colors.append({
                'hex': color_element['data-hex'].upper(),
                'rgb': get_color_data(color_element['data-rgb'], rgb_keys),
                'hsl': get_color_data(color_element['data-hsl'], hsl_keys)
            })

    return colors
```

- Metoda **parse_colors** are rolul să parcurgă elementele HTML și să extragă informațiile necesare din acestea.
- În imaginea de mai jos puteți vedea structura elementelor care ne interesează:

```
▼<div class="color-group">
  ▼<div class="color-block">
     <div class="js-color" style="background-color: #f5eef8" data-hex=
     "#f5eef8" data-rgb="245, 238, 248" data-hsl="283, 39%, 95%"></div>
   </div>
```

- Cu alte cuvinte, vom căuta informația noastră în atributele **data-hex**, **data-rgb** și **data-hsl** ale fiecărui **div** ce are clasa **js-color** și se află într-un **div** ce are clasa **color-group**.
- Toate aceste elemente se află într-un element **header** care are ID-ul **flat**.

Codebase

```python
def get_values_from_string(data):
    result = []

    pattern = r'[0-9]+%?'
    matches = re.findall(pattern, data)

    for match in matches:
        result.append(match)

    return result


def get_color_data(string, keys):
    values = get_values_from_string(string)
    return {key: value for key, value in zip(keys, values)}
```

- Metoda **get_color_data** primește un string care conține 3 valori separate de virgulă și un tuplu de forma **('r', 'g', 'b')** sau **('h', 's', 'l')**.
- String-ul este trimis ca parametru metodei **get_values_from_string** care parsează string-ul folosind regex pentru a obține valorile RGB, respectiv HSL.
- Metoda **get_color_data** returnează un dicționar pe baza valorilor din string și a cheilor, astfel:

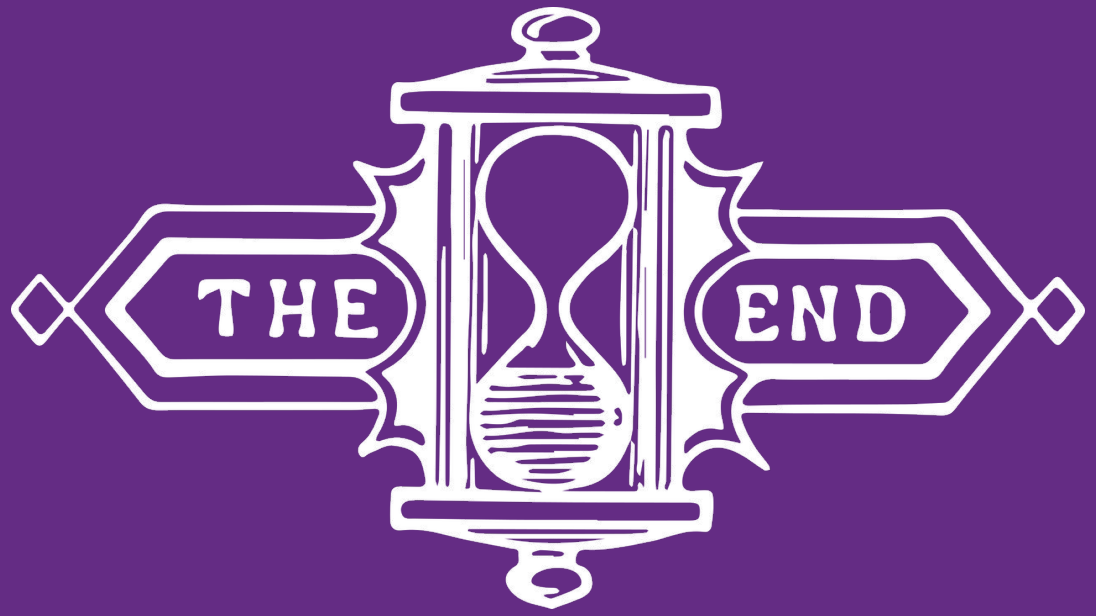
```
"rgb": {
    "r": "249",
    "g": "235",
    "b": "234"
},
```

```
"hsl": {
    "h": "6",
    "s": "54%",
    "l": "89%"
}
```

Codebase

```python
def get_values_from_string(data):
    result = []

    pattern = r'[0-9]+%?'
    matches = re.findall(pattern, data)

    for match in matches:
        result.append(match)

    return result


def get_color_data(string, keys):
    values = get_values_from_string(string)
    return {key: value for key, value in zip(keys, values)}
```

- Metoda **get_values_from_string** primește string-ul care conține valorile **RGB** sau **HSL** separate prin virgulă astfel:
  - **RGB:** 249, 235, 234
  - **HSL:** 6, 54%, 95%
- Pattern-ul pentru identificarea acestor valori este format din:
  - **[0-9]+** - orice cifră **[0-9]** gasită o dată sau de mai multe ori **+**
  - **%?** - caracterul **%** găsit de zero ori sau maxim o dată **?**
- După identificarea tuturor aparițiilor vom parcurge lista obținută și vom crea o listă cu valorile obținute.
- Acestea sunt folosite în metoda **get_color_data** pentru a crea dicționarul dorit.



Vă mulțumesc!