Oppgave 1:

The wikipedia page for Ukraine (https://en.wikipedia.org/wiki/Ukraine) contains several images and captions arranged inside divs with class name "thumbinner". A screenshot of one of the images and its corresponding HTML tags are shown in the following figure. The tag inside

contains a link to the image; and the child

contains image captions.

In this task you have to scrape all the image links and captions those are inside divs with class name "thumbinner". The output of your program should be the following. Note that the output of the caption should not include any links but only plain text.

Answer:

import re  
from urllib.parse import urljoin  
from urllib.request import urlopen  
from bs4 import BeautifulSoup  
  
html = urlopen("https://en.wikipedia.org/wiki/Ukraine")  
bs = BeautifulSoup(html.read(), "html.parser")  
start\_url = "https://en.wikipedia.org/wiki/"  
  
thumbinner\_divs = bs.find\_all("div", {'class': "thumbinner"})  
  
for div in thumbinner\_divs:  
 image\_finder = div.find("img")  
 if image\_finder:  
 img\_source = image\_finder.get("src")  
  
 caption\_finder = div.find("div", {'class':"thumbcaption"}).text  
  
 print(img\_source)  
 print(caption\_finder)  
  
"""Task for finding image links and the caption of each image and then printing it using bs4."""

Oppgave 2:

Task: Write a python program that scrapes all the images from https://www.pythonscraping.com/pages/page3.html. The program should filter the product images. There are two logo images at the top those have different pattern. Your program should only print out the links to the product images.

Answer:

from selenium import webdriver  
from selenium.webdriver.common.by import By  
from selenium.webdriver.support.ui import WebDriverWait  
from selenium.webdriver.support import expected\_conditions as EC  
import random  
import time  
from urllib.parse import urlsplit  
  
driver = webdriver.Chrome()  
start\_url = "https://www.pythonscraping.com/pages/page3.html"  
driver.get(start\_url)  
time.sleep(3)  
  
wait = WebDriverWait(driver, 10)  
table = wait.until(EC.presence\_of\_element\_located((By.XPATH, "//table")))  
  
elements = driver.find\_elements(By.XPATH,'//table//img')  
print(elements)  
for element in elements:  
 partial\_link = urlsplit(element.get\_attribute('src'))  
 print(partial\_link[2])  
  
driver.quit()

Oppgave 3.

What is Model-View-Controller (MVC) framework ? Write briefly about few benefits of using MVC.

Answer:

A screenshot of a computer program

Description automatically generated with low confidence

Oppgave 4. ‘

Task: Following is a SPARQL query that fetches data from Wikidata.org and displays the results in a world map. The output of the query is a map consisting of all the hospitals in the world. Your task is to update the query so that it shows only the hospitals in Western Europe. You may use the 'country' property of hospitals and 'part of' property of countries to specify the query. --------------------------------#Map of hospitals

#defaultView:Map

SELECT distinct \* WHERE { ?item wdt:P31/wdt:P279\* wd:Q16917;

wdt:P625 ?geo . }

wdt:P31 -> instance of

wdt:P279 -> subclass of

wd:Q16917 -> Hospital

wdt: P625 -> coordinate location

wdt:P17 -> country

wdt:P361 -> part of

wd:Q27496 -> Western Europe

answer:

SELECT DISTINCT ?hospital ?hospitalLabel ?geo WHERE {

# Instance of hospital

?hospital wdt:P31/wdt:P279 wd:Q16917.

?hospital wdt:P17/wdt:P361 wd:Q27496.

?hospital wdt:P625 ?geo.

}

Oppgave 5.

Task: Load Zachary's karate club graph using the following networkx library:

G = nx.karate\_club\_graph()

and apply the following social network analysis measure to the Karate club graph:

• Measure centrality based on degree centrality using networkx library.

• Detect modularity of the graph using label propagation or louvain method (use networkx

libraries).

• Modify the size based on degree centrality and color based on the modularity measures.

• Use spring layout or fruchterman\_reingold layout for visualizing the graph.

Answer:

import networkx as nx  
from networkx import community  
from networkx.algorithms.community import greedy\_modularity\_communities, label\_propagation\_communities  
  
G = nx.karate\_club\_graph()  
#Measuring centrality using degree centrality  
degree\_centrality = nx.degree\_centrality(G)  
  
#measuring modularity with louvain method and label propagation  
#modularity first;  
communities = greedy\_modularity\_communities(G)  
partitions = {}  
for i, comm in enumerate(communities):  
 for node in comm:  
 partitions[node] = i  
  
#Measuring modularity:  
louvain\_modularity = nx.algorithms.community.modularity(G, communities)  
  
"""the modularity score tells how strongly nodes are connected within the communities.  
worse than 0 means that the network is highly fragmented, the closer the score is to 1 the more connected nodes within  
communities are."""  
#label propagation  
propagation\_communites = label\_propagation\_communities(G)  
for i, comm in enumerate(propagation\_communites):  
 print("communities", i, comm, "\n")  
 """This gives a list of the nodes that belong to the same score, and not a modularity score.  
 but it gives a nice overview over the communities."""  
  
print("centrality of the nodes", degree\_centrality)  
print("modularity using lovuain", louvain\_modularity)  
  
#changing size and color using degree centrality and modularity:  
#layout:  
pos = nx.spring\_layout(G)  
  
#changing colors:  
node\_colors = {}  
for node in G.nodes:  
 node\_colors[node] = node  
  
#changing node size:  
node\_sizes=[]  
for node in G.nodes:  
 node\_sizes.append(degree\_centrality[node] \* 1500)  
  
#drawing graph:  
import matplotlib.pyplot as plt  
  
# Draw nodes  
nx.draw\_networkx\_nodes(G, pos, node\_color=list(node\_colors.values()), node\_size=node\_sizes, cmap='cool')  
  
# Draw edges  
nx.draw\_networkx\_edges(G, pos)  
# Draw labels (optional)  
nx.draw\_networkx\_labels(G, pos)  
  
# Show the graph  
plt.axis('off')  
plt.show()

BS4 koding  
task: beskrevet I koden:

import urllib.request  
from urllib.parse import urljoin, urlparse  
import requests  
from bs4 import BeautifulSoup  
import random  
import re  
  
url = "https://simple.wikipedia.org/wiki/Abbey\_Road"  
response = urllib.request.urlopen(url)  
html = response.read()  
bs = BeautifulSoup(html, "html.parser")  
  
#TASK1 find all hrefs attributes found in a tags.  
def task1():  
 try:  
 links = bs.find\_all("a")  
 for link in links:  
 link = link.get("href")  
 print(link)  
 except IndexError as e:  
 pass  
#task1()  
  
#TASK2 make a function that only find internal links(/wiki/)  
def task2():  
 base\_url = "https://simple.wikipedia.org"  
 try:  
 links = bs.find\_all("a")  
 for link in links:  
 link = link.get("href")  
 if link.startswith("/wiki"):  
 print(link)  
 except IndexError as e:  
 pass  
#task2()  
  
  
#TASK3 #make the relative links from task 2 into absolute links  
def task3():  
 base\_url = "https://simple.wikipedia.org"  
 try:  
 links = bs.find\_all("a")  
 for link in links:  
 link = link.get("href")  
 if link is not None and link.startswith("/wiki/"):  
 full\_link = urljoin(base\_url, link)  
 print(full\_link)  
 else:  
 continue  
 except IndexError as e:  
 pass  
#task3()  
  
  
#TASK4 write a function that selects 10 random absolute links from task 3. and visit them print visited links and heading of the link  
  
def task4():  
 base\_url = "https://simple.wikipedia.org"  
 absolute\_links = []  
 try:  
 links = bs.find\_all("a")  
 for link in links:  
 link = link.get("href")  
 if link is not None and link.startswith("/wiki/"):  
 full\_link = urljoin(base\_url, link)  
 absolute\_links.append(full\_link)  
 else:  
 continue  
 except IndexError as e:  
 pass  
  
 counter = 0  
 while counter <= 10:  
 random\_links\_visit = random.sample(absolute\_links, 10)  
 for random\_link in random\_links\_visit:  
 if counter > 9:  
 break  
 else:  
 counter +=1  
 print(random\_link)  
 new\_response = urllib.request.urlopen(random\_link)  
 soup = BeautifulSoup(new\_response, "html.parser")  
 header = soup.find("h1").text  
 print(header)  
 print(counter)  
#task4()  
  
#TASK 5 function for crawling all albums and printing album duration and release year.  
def task5():  
 url = "https://simple.wikipedia.org/wiki/Category:The\_Beatles\_albums"  
 base\_url = "https://simple.wikipedia.org"  
 response = urllib.request.urlopen(url)  
 html = response.read()  
  
 bsoup = BeautifulSoup(html, "html.parser")  
 albums\_location = bsoup.find("div", attrs={"class":"mw-category-generated"})  
 links = albums\_location.find\_all("a")  
  
 album\_list = []  
 for link in links:  
 link = link.get("href")  
 link = urljoin(base\_url, link)  
 album\_list.append(link)  
  
 for album in album\_list:  
 while len(album\_list) == 0:  
 break  
 else:  
 new\_response = urllib.request.urlopen(album)  
 html\_new = response.read  
 b\_s = BeautifulSoup(html\_new, "html.parser")  
  
 release\_date = b\_s.find("td", attrs={"class":"infobox-data category"})  
 print(release\_date)  
  
task5()

Mer bs4 med litt regex:

import urllib.request  
from urllib.request import urlopen  
from urllib.parse import urljoin, urlparse  
from bs4 import BeautifulSoup  
import re  
  
# Open the URL  
url = "https://simple.wikipedia.org/wiki/Abbey\_Road"  
response = urllib.request.urlopen(url)  
html = response.read()  
  
bs = BeautifulSoup(html, "html.parser")  
  
#TASK1 finn lengden på albumet  
album\_tables = bs.find("span", attrs={"class":"duration"}).text  
print(album\_tables)  
  
#TASK4  
#find and print titles of songs from the first page of the album:  
titles\_list = []  
table = bs.find("table", attrs={"class":"tracklist"})  
titles = table.find\_all("a")  
for title in titles:  
 titles\_list.append(title.get\_text())  
print(titles\_list)  
  
#TASK5  
#find all a tags within paragraphs(p) use try, except.  
base\_url = url  
paragraphs = bs.find\_all("p")  
hrefs = []  
for p in paragraphs:  
 try:  
 links = p.find\_all("a")  
 for link in links:  
 href = link.get("href")  
 full\_href = urljoin(base\_url, href)  
 hrefs.append(full\_href)  
 except AttributeError:  
 print("no hrefs")  
print(hrefs)  
  
  
#TASK6  
# Task 6 - Print all Songs with lead vocals by Mccartney  
def task6():  
 songs\_Mcartney = []  
 print("\nTask 6: ")  
 for table in bs.find\_all("table", attrs={"class":"tracklist"}):  
 for title in table.find\_all("tr"):  
 rows = title.find\_all("td")  
 # I run Exception handling here because some of the table rows  
 # had inconsistent format, and there gave errors when I tried to get the song title or vocalist.  
 try:  
 if rows[2].text == "McCartney":  
 songs\_Mcartney.append(rows[1].text)  
 print(songs\_Mcartney)  
 except IndexError as e:  
 pass  
  
task6()  
  
#TASK7  
#REGEX  
links = bs.find\_all("a", href=re.compile("^(?!/you/)$"))  
def task7():  
 print("\nTask 7: ")  
 for table in bs.find\_all("table", attrs={"class":"tracklist"}):  
 for title in table.find\_all("tr"):  
 rows = title.find\_all("td")  
 try:  
 # Regex search for the word "You" or "you".  
 result = re.search("^.\*[Yy]ou.\*$", rows[1].text)  
 if result != None:  
 print(result[0])  
 except IndexError as e:  
 pass  
  
task7()

Oppg: spacy med newspaper3k

import urllib.request  
import re  
from urllib.parse import urljoin  
import csv  
from newspaper import Article  
import spacy  
import nltk  
from collections import Counter  
from bs4 import BeautifulSoup  
  
nlp = spacy.load("en\_core\_web\_sm")  
url = "https://simple.wikipedia.org/wiki/The\_Hobbit"  
response = urllib.request.urlopen(url)  
html = response.read()  
  
bs = BeautifulSoup(html, "html.parser")  
  
#collecting text from all paragraphs into one string.  
doc = ""  
for p in bs.find\_all("p"):  
 doc += p.text  
doc = nlp(doc)  
  
#TASK1 print all tokens from the string  
def find\_tokens():  
 for token in doc:  
 print(token.text)  
find\_tokens()  
  
#TASK2 find nouns  
def find\_nouns():  
 for token in doc:  
 if token.pos\_ == "NOUN":  
 print(token)  
find\_nouns()  
  
#TASK3 find enteties  
def find\_enteties():  
 for ent in doc.ents:  
 print(ent.text)  
find\_enteties()  
  
#TASK4 print the 10 most mentioned enteties  
def most\_mentioned\_enteties():  
 unique\_ents = []  
 for ent in doc.ents:  
 unique\_ents.append(ent.text)  
 frequencies = Counter([ent.text for ent in doc.ents])  
 print(frequencies.most\_common(10))  
  
most\_mentioned\_enteties()

**Mer teori:**

OSI model (open systems interconnection)

The Open Systems Interconnection (OSI) model is a conceptual model created by the

International Organization for Standardization which enables diverse communication systems to communicate using standard protocols.

• The OSI provides a standard for different computer systems to be able to

communicate with each other.

• The OSI model organizes computer communication into seven layers:

• Layer7: Application Layer: HTTP, FTP and DNS, for instance.

• Layer6: Presentation Layer: Includes protocols to format and translate data. ASCII, SSL, for instance

• Layer5: Session Layer: Includes protocols for opening/closing and managing sessions.

• Layer4: Transport Layer: TCP, but also protocols such as UDP, which do not offer the advanced error checking and recovery mechanisms of TCP for sake of speed.

• Layer3: Network Layer: Includes IP (Internet Protocol).

• Layer2: Data link Layer: Includes the Ethernet protocol.

• Layer1: Physical Layer: Includes the Ethernet protocol, but also USB, Bluetooth, and other radio protocols

Hypertext transfer protocol(HTTP)

A Web page consist of objects

Object can be HTML file, image, audio clip, video clip, java applet

A web page consists of a base HTML file which includes a collection

of referenced objects

Each object is accessed by a Uniform Resource Locator (URL)

**Scrapy:**

*Scrapy Engine* The engine is responsible for controlling the data flow between all components of the system, and triggering events when certain actions occur.

*Spiders* are custom classes written by Scrapy users to parse responses and extract items from them or additional requests to follow

*The Scheduler* receives requests from the engine and enqueues them for feeding them later (also to the engine) when the engine requests them.

*The Downloader* is responsible for fetching web pages and feeding them to the engine which, in turn, feeds them to the spiders.

*The Item Pipeline* is responsible for processing the items once they have been extracted (or scraped) by the spiders. Typical tasks include cleansing, validation and persistence (like storing the item in a database).

*Word embedding*, in NLP, is a representation of the meaning of words.

*Token*: A word, punctuation mark etc. in context, including its attributes, tags and dependencies.

*Lexeme*: A “word type” with no context. Includes the word shape and flags, e.g. if it’s lowercase, a digit or punctuation.

*Doc*: A processed container of tokens in context.

*Vocab*: The collection of lexemes.

*StringStore*: The dictionary mapping hash values to  
  
n-Gram recurring set of words often used to getter. A function implementing n-gram requires data cleaning.