Scrapping

1. **Leer un tipo text(mucho texto)**

```
sinopsis = s.find("div", class_="info").get_text #Sin string ni strip
```

2. **Leer texto dentro de un contenedor con una etiqueta previa al texto(línea 52 examen scrapping)**

```
hora = s1.find("div", class_="hora").find("div", class_="value")
                for c in hora.children:
                    if c.name != "i":
                        horario = c.strip()
                horario = parse_horario(horario)
```

3. **Leer un dato que puede estar vacío (comprobamos el if con el contenedor anterior al que buscamos)**

```
if s1.find("div", class_="lugar"):
lugar=s1.find("div",class_="lugar").find("div",class_="value").p.string.strip(
)
else:
       lugar = NULL
```

4. **Parseo de fecha básico**

```
def parse_fechas(fecha):
    res = []
    if "al" in fecha:
        fechas = fecha.split("al")
        fecha_inicio = datetime.strptime(fechas[0].strip(),'%d/%m/%Y')
        fecha_fin = datetime.strptime(fechas[1].strip(), '%d/%m/%Y')
        res.append(fecha_inicio)
        res.append(fecha_fin)

    else:
        fecha = datetime.strptime(fecha.strip(), '%d/%m/%Y')
        res.append(fecha)
        res.append(fecha)

    return res
```

5. **Parseo de tiempo básico**

```
def parse_horario(horario):
    if 'cena' in horario:
        horario = time(hour=20, minute=0)
    elif len(horario.split(" "))>1:
        horario = horario.split(" ")[-1].strip()
        horas = horario[0].split(":")[0].strip()
        minutos = horario[1].split(":")[0].strip()
        horario = time(hour=int(horas), minute=int(minutos))
    else:
        horas = horario[0].split(":")[0].strip()
        minutos = horario[1].split(":")[0].strip()
        horario = time(hour=int(horas), minute=int(minutos))
    return horario.strftime("%I:%M %p")
```

6. **Buscar por atributo especial (Si hay varios contenedores con la misma clase)**

```
complejidad = t.find("div",attrs={"data-th":"Complejidad"})
```

Almacenar con Woosh

1. Función almacenar base

```
def almacenar_datos():
    #1
      schem = Schema(titulo=TEXT(stored=True,phrase=False),
      precio=NUMERIC(stored=True,numtype=float),
      tematicas=KEYWORD(stored=True,commas=True,lowercase=True),
      complejidad=ID(stored=True),
      jugadores=KEYWORD(stored=True,commas=True), detalles=TEXT)

    #2
    if os.path.exists("Index"):
        shutil.rmtree("Index")
    os.mkdir("Index")

    #3
    ix = create_in("Index", schema=schem)
    writer = ix.writer()
    i=0
    lista=extraer_juegos()
    for j in lista:
    #4
      writer.add_document(titulo=str(j[0]), precio=float(str(j[1])),
      tematicas=str(j[2]), complejidad=str(j[3]), jugadores=str(j[4]),
      detalles=str(j[5]))
       i+=1
    writer.commit()
    messagebox.showinfo("Fin de indexado", "Se han indexado "+str(i)"
juegos")
```

#1: creamos el schema donde le damos a cada variable que vayamos a utilizar un tipo y atributos

#2: Si existe el path del índice lo borra y crea uno nuevo

#3: Solo habría que cambiar lo subrayado por nuestra función de extraer

#4: Ponemos las variables de que extraemos como listas y su tipo, CUIDADO con los datetime y puede que algún tipo más, no hay que transformarlo, simplemente variable=j[x]

2. **Tipos de Schema**

a. Stored=True, hace que se almacene la variable para mostrarla en algún momento, por lo que si no nos pone nada de que no se pueda mostrar la ponemos en todas.

b. Si para la búsqueda hay que escribir la palabra o frase entera, (phrase=False)
`titulo=TEXT(stored=True,phrase=True)`

c. Si para la búsqueda hay que escribir una palabra que esté contenida en un texto o string,(phrase=False), ya que esto guarda cada palabra separada por , o espacio, es útil para buscar los text que contengan una palabra.
`titulo=TEXT(stored=True,phrase=False)`

d. Si queremos almacenar palabras separadas por comas (como podría ser el caso de las temáticas: Terror, Ciencia Ficción), utilizamos el tipo Keyword y commas=true(lowercase nos quita de problemas).
`tematicas=KEYWORD(stored=True,commas=True,lowercase=True)`

e. Si queremos almacenar palabras separadas por comas (como podría ser el caso de las temáticas: Terror, Ciencia Ficción), utilizamos el tipo Keyword y commas=true.

f. El tipo ID, debería de usarse para valores únicos, pero en woosh solo indica que está formado por una palabra o cadena de números
`url_peli=ID(stored=True, unique=True)`

g. Cuando queramos modificar o eliminar valor, este obligatoriamente debe ser de tipo ID y con el atributo unique=True
`url_peli=ID(stored=True, unique=True)`

h. Los tipos numéricos se representan así
`precio=NUMERIC(stored=True,numtype=float)`

i. **IMPORTANTE** los tipos NUMERIC, DATETIME, etc. Hay que importarlos al inicio del documento, abajo pongo un import con todo lo que he usado para copiar y pegar

```
#encoding:utf-8

from bs4 import BeautifulSoup
import urllib.request
from tkinter import *
from tkinter import messagebox
import sqlite3
import lxml
import re, shutil
from whoosh import qparser, query
from whoosh.index import create_in,open_dir
from whoosh.fields import Schema, TEXT, NUMERIC, KEYWORD, ID, DATETIME
from whoosh.qparser import QueryParser, OrGroup
from whoosh.query import Term, Or

from datetime import datetime
# lineas para evitar error SSL
import os, ssl
from _datetime import date
if (not os.environ.get('PYTHONHTTPSVERIFY', '') and
getattr(ssl, '_create_unverified_context', None)):
```

```
ssl._create_default_https_context = ssl._create_unverified_context
```

Consultas tkinter

1. **Imprimir_lista()**

```
def imprimir_lista(cursor):
    v = Toplevel()
    v.title("Películas el séptimo arte")
    sc = Scrollbar(v)
    sc.pack(side=RIGHT, fill=Y)
    lb = Listbox(v, width = 150, yscrollcommand=sc.set)
    for row in cursor:
        lb.insert(END,row['titulo'])
        lb.insert(END,row['titulo_original'])
        lb.insert(END,row['fecha'])
        lb.insert(END,row['pais'])
        lb.insert(END,row['generos'])
        lb.insert(END,row['director'])
        lb.insert(END,row['sinopsis'])
        lb.insert(END,row['url_peli'])
        lb.insert(END,"\n\n")
    lb.pack(side=LEFT,fill=BOTH)
    sc.config(command = lb.yview)
```

Implemente creamos un insert por cada atributo que queramos mostrar

2. **Listar_todo()**

```
def listar_todo():
        ix = open_dir("Index")
        with ix.searcher() as searcher:
            results = searcher.search(query.Every(),limit=None)
            imprimir_lista(results)
```

Función típica para mostrar todos los resultados, **IMPORTANTE** hay que importar query en woosh (Si copias el encabezado que puse ya viene importado)

3. **Buscar por un atributo u otro**

```
def buscar_titulo_sinopsis():
    def mostrar_lista(event):
        #abrimos el Ãndice
        ix=open_dir("Index")
        #creamos un searcher en el Ãndice
        with ix.searcher() as searcher:
            query = MultifieldParser(["titulo","sinopsis"],
ix.schema, group=OrGroup).parse(str(en.get()))
            results = searcher.search(query)
            #recorremos los resultados obtenidos(es una lista de
diccionarios) y mostramos lo solicitado
            v = Toplevel()
            v.title("Listado de Peliculas")
            v.geometry('800x150')
            sc = Scrollbar(v)
            sc.pack(side=RIGHT, fill=Y)
            lb = Listbox(v, yscrollcommand=sc.set)
            lb.pack(side=BOTTOM, fill = BOTH)
            sc.config(command = lb.yview)
            #Importante: el diccionario solo contiene los campos
que han sido almacenados(stored=True) en el Schema
            for r in results:
                lb.insert(END,r['titulo'])
                lb.insert(END,r['titulo_original'])
                lb.insert(END,r['director'])
                lb.insert(END,'')

    v = Toplevel()
    v.title("Busqueda por TÃtulo o Sinopsis")
    l = Label(v, text="Introduzca las palabras a buscar:")
    l.pack(side=LEFT)
    en = Entry(v)
    en.bind("<Return>", mostrar_lista)
    en.pack(side=LEFT)
```

1:

`MultifieldParser(["titulo","sinopsis"],` Si queremos buscar entre dos atributos hay que usar el MultifieldParser, después metemos en la lista los atributos entre los que vamos a filtrar.
`group=OrGroup`, Aquí indicamos que la condición será OR, para que sea AND, ponemos AndGroup

2:

Aquí simplemente ponemos los datos que queremos que nos muestre al hacer la búsqueda

#Además de eso hay que cambiar los títulos y si nos piden otro tipo de muestreo de los datos chatgpt

4. **Buscar por un atributo**

```
# permite buscar las pelÃculas de un "gÃ©nero"
def buscar_generos():
    def mostrar_lista(event):
        ix=open_dir("Index")
        with ix.searcher() as searcher:
            lista_generos = [i.decode('utf-8') for i in
searcher.lexicon('generos')]
            entrada = str(en.get().lower())

        if entrada not in lista_generos:
                messagebox.showinfo("Error", "El criterio de bÃºsqueda
no es un gÃ©nero existente\nLos gÃ©neros existentes son: " +
",".join(lista_generos))
                return

            query = QueryParser("generos",
ix.schema).parse('"'+entrada+'"')
            results = searcher.search(query, limit=20)

            v = Toplevel()
            v.title("Listado de PelÃculas")
            v.geometry('800x150')
            sc = Scrollbar(v)
            sc.pack(side=RIGHT, fill=Y)
            lb = Listbox(v, yscrollcommand=sc.set)
            lb.pack(side=BOTTOM, fill = BOTH)
            sc.config(command = lb.yview)
            for r in results:
                lb.insert(END,r['titulo'])
                lb.insert(END,r['titulo_original'])
                lb.insert(END,r['pais'])
                lb.insert(END,'')

    v = Toplevel()
    v.title("Busqueda por GÃ©nero")
    l = Label(v, text="Introduzca gÃ©nero a buscar:")
    l.pack(side=LEFT)
    en = Entry(v)
    en.bind("<Return>", mostrar_lista)
    en.pack(side=LEFT)
```

1:

Cargas todos los géneros, solo hay que cambiar el searcher

2:

Cargas la entrada en minúscula para que no te de fallo de coincidencia

3:

Igual que el anterior, solo que en vez de usar MultifieldParser, usa QueryParser, ya que es solo un atributo para filtrar

5. **Modificar o eliminar una entrada (Debe ser tipo ID con unique=True)**

```
def modificar_fecha():
    def modificar():
        #comprobamos el formato de la entrada
        if(not re.match("\d{8}",en1.get())):
            messagebox.showinfo("Error", "Formato del rango de fecha incorrecto")
            return
        ix=open_dir("Index")
        lista=[]
        with ix.searcher() as searcher:
            query = QueryParser("titulo", ix.schema).parse(str(en.get()))
            results = searcher.search(query, limit=None)

            v = Toplevel()
            v.title("Listado de PelÃculas a Modificar")
            v.geometry('800x150')
            sc = Scrollbar(v)
            sc.pack(side=RIGHT, fill=Y)
            lb = Listbox(v, yscrollcommand=sc.set)
            lb.pack(side=BOTTOM, fill = BOTH)
            sc.config(command = lb.yview)

            for r in results:
                lb.insert(END,r['titulo'])
                lb.insert(END,r['fecha'])
                lb.insert(END,'')
                lista.append(r) #cargamos la lista con los resultados de la bÃºsqueda
        # actualizamos con la nueva fecha de estreno todas las pelÃculas de la lista
        respuesta = messagebox.askyesno(title="Confirmar",message="Esta seguro que
quiere modificar las fechas de estrenos de estas peliculas?")
        if respuesta:
            writer = ix.writer()
            for r in lista:
                writer.update_document(url=r['url'],
fecha=datetime.strptime(str(en1.get()),'%Y%m%d'), titulo=r['titulo'],
titulo_original=r['titulo_original'], pais=r['pais'], director=r['director'],
generos=r['generos'], sinopsis=r['sinopsis'])
            writer.commit()

    v = Toplevel()
    v.title("Modificar Fecha Estreno")
    l = Label(v, text="Introduzca TÃtulo PelÃcula:")
    l.pack(side=LEFT)
    en = Entry(v)
    en.pack(side=LEFT)
    l1 = Label(v, text="Introduzca Fecha Estreno AAAAMMDD:")
    l1.pack(side=LEFT)
    en1 = Entry(v)
    en1.pack(side=LEFT)
    bt = Button(v, text='Modificar', command=modificar)
    bt.pack(side=LEFT)
```

1:

Ahí ponemos el atributo por el que filtraremos en este caso el título

2:

Aquí llamamos al método de actualizar entrada, el primer valor debe ser el que hemos denotado como ID (unique=True). Para eliminar sería `writer.delete_by_term("url", r['url'])`

3:

Dejamos todos los datos de la lista iguales excepto el que queremos modificar, al que le ponemos el valor de la entrada

6. Buscar con rango

```
def buscar_fecha():
def mostrar_lista(event):
    #comprobamos el formato de la entrada
    if(not re.match("\d{8}\s+\d{8}",en.get())):
        messagebox.showinfo("Error", "Formato del rango de fecha incorrecto")
        return
    ix=open_dir("Index")
    with ix.searcher() as searcher:

        aux = en.get().split()
        rango_fecha = '['+ aux[0] + ' TO ' + aux[1] +']'
        query = QueryParser("fecha", ix.schema).parse(rango_fecha)
        results = searcher.search(query,limit=None) #devuelve todos los resultados
        v = Toplevel()
        v.title("Listado de PelÃculas")
        v.geometry('800x150')
        sc = Scrollbar(v)
        sc.pack(side=RIGHT, fill=Y)
        lb = Listbox(v, yscrollcommand=sc.set)
        lb.pack(side=BOTTOM, fill = BOTH)
        sc.config(command = lb.yview)
        for r in results:
            lb.insert(END,r['titulo'])
            lb.insert(END,r['fecha'])
            lb.insert(END,'')

v = Toplevel()
v.title("Busqueda por Fecha")
l = Label(v, text="Introduzca rango de fechas AAAAMMDD AAAAMMDD:")
l.pack(side=LEFT)
en = Entry(v)
en.bind("<Return>", mostrar_lista)
en.pack(side=LEFT)
```

1:

Creamos el rango introduciendo la entrada separada por TO

2:

Pasamos el parseo a la query y este mismo hace el filtro

Versión más cómoda:

```
query = NumericRange("fecha", fecha_limite, None)
```

(Atributo, inicio, fin)