UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

"WEB SCRAPING: IL CASO DI STUDIO DEL VOCABOLARIO DEL COMMERCIO MEDIEVALE"

Relatore: Prof. / Dott. GIORGIO MARIA DI NUNZIO

Laureando/a: FRANCESCO CHEMELLO
Matricola n° 1219613

ANNO ACCADEMICO 2022 – 2023

Indice

Introduzione

L'obiettivo di questa tesi di laurea è quello di presentare un programma Python di *web scraping* che estrapola i dati linguistici contenuti nel sito *"https://www.um.es/lexico-comercio-medieval/index.php"*, li elabora e li salva in un foglio elettronico (*.xlsx*) che potrà poi esser utilizzato come base per la realizzazione di un database terminologico.

In questo scritto si vogliono illustrare tutti i procedimenti che hanno portato alla sua realizzazione soffermandosi sull'analisi del contesto di lavoro, sulle modalità di progettazione, sviluppo e distribuzione e sul suo funzionamento nel dettaglio.

Il documento si articola in *quattro capitoli*:

- Il *primo capitolo* corrisponde ad una breve panoramica del sito.
- Il *secondo capitolo* espone il procedimento di risoluzione del problema, le tecniche usate per lo sviluppo del software e la sua distribuzione.
- Il *terzo capitolo* analizza in modo approfondito la struttura ed il funzionamento del programma.
- Il *quarto capitolo* riporta in dettaglio le funzioni che il programma usa.

Capitolo 1 - Vocabolario del Commercio Medievale

1.1 – Introduzione

In questo capitolo viene presentato il *Vocabulario del comercio medieval* soffermandosi sul suo contenuto e sulla sua realizzazione.

1.2 – Contenuto

Il vocabolario del commercio medievale, consultabile alla pagina "*https://www.um.es/lexico-comercio-medieval/index.php*", è un database dell'Università spagnola di Murcia di fonti medievali che documentano, definiscono e descrivono prodotti commerciali, scambi, tasse, pesos, misure, monete ed istituzioni del periodo compreso tra il IX secolo e l'XVI secolo. Quest'opera è frutto dell'unione di più fonti che sono:

- *Vocabulario del comercio medieval* di Miguel Gual Camarena.
- *Manual de mercadería* di Miguel Gual Camarena.
- Gli scritti inediti di Miguel Gual Camarena.
- *Materiales para un diccionario de Historia Económica Hispana (siglo IX-XVI)* frutto di una borsa di studio della Fondazione Juan March[1].

Tutte le voci sono consultabili attraverso l'utilizzo di una barra di ricerca presso la pagina *"https://www.um.es/lexico-comercio-medieval/index.php/vocabulario"*.

1.3 - Realizzazione

Il vocabolario del commercio medievale deve in gran parte la sua esistenza al lavoro di una vita del professore di storia medievale Miguel Gual Camarena, discepolo di Vicens Vives[2]. Suo figlio José Miguel, vista l'importanza e l'unicità del lavoro di suo padre, decise di digitalizzare tutti i suoi scritti e, nel 2013, li donò all'Università di Murcia perché venissero resi disponibili in formato digitale a tutta la comunità scientifica.

[1] *La Fondazione Juan March è nata nel 1955 dall'omonimo uomo d'affari e banchiere spagnolo con la missione di promuovere la cultura in Spagna.*
[2] *1910-1960, storico spagnolo e professore di storia dell'Università di Saragozza*

Vista l'enorme mole di dati da elaborare, il lavoro è ancora in corso e si prevede che verrà completato entro il 2030.

Il processo di realizzazione è suddiviso in quattro fasi:

- La *prima fase* (nel 2013) è stata la digitalizzazione di tutti i lavori di Gual Camarena, conservati dalla sua famiglia e donati all'Università di Murcia.
- La *seconda fase* (nel 2014) è stata la digitalizzazione di tutti i lavori frutto della Fondazione Juan March da parte di trenta ricercatori delle Università di Granada, Murcia, Malaga, Navarra, Maiorca, Valencia, Salamanca, Siviglia e Barcellona.
- La *terza fase* (tra il 2015 e il 2022) ha permesso la dattilografia dei file manoscritti del Prof. Gual e quelli della borsa di studio Juan March.
- La *quarta fase* (iniziata nel 2022 e con termine previsto nel 2027) comprende la revisione di tutti i materiali, la scrittura delle voci e l'incorporazione della nuova bibliografia.

Capitolo 2 - Sviluppo software e pubblicazione

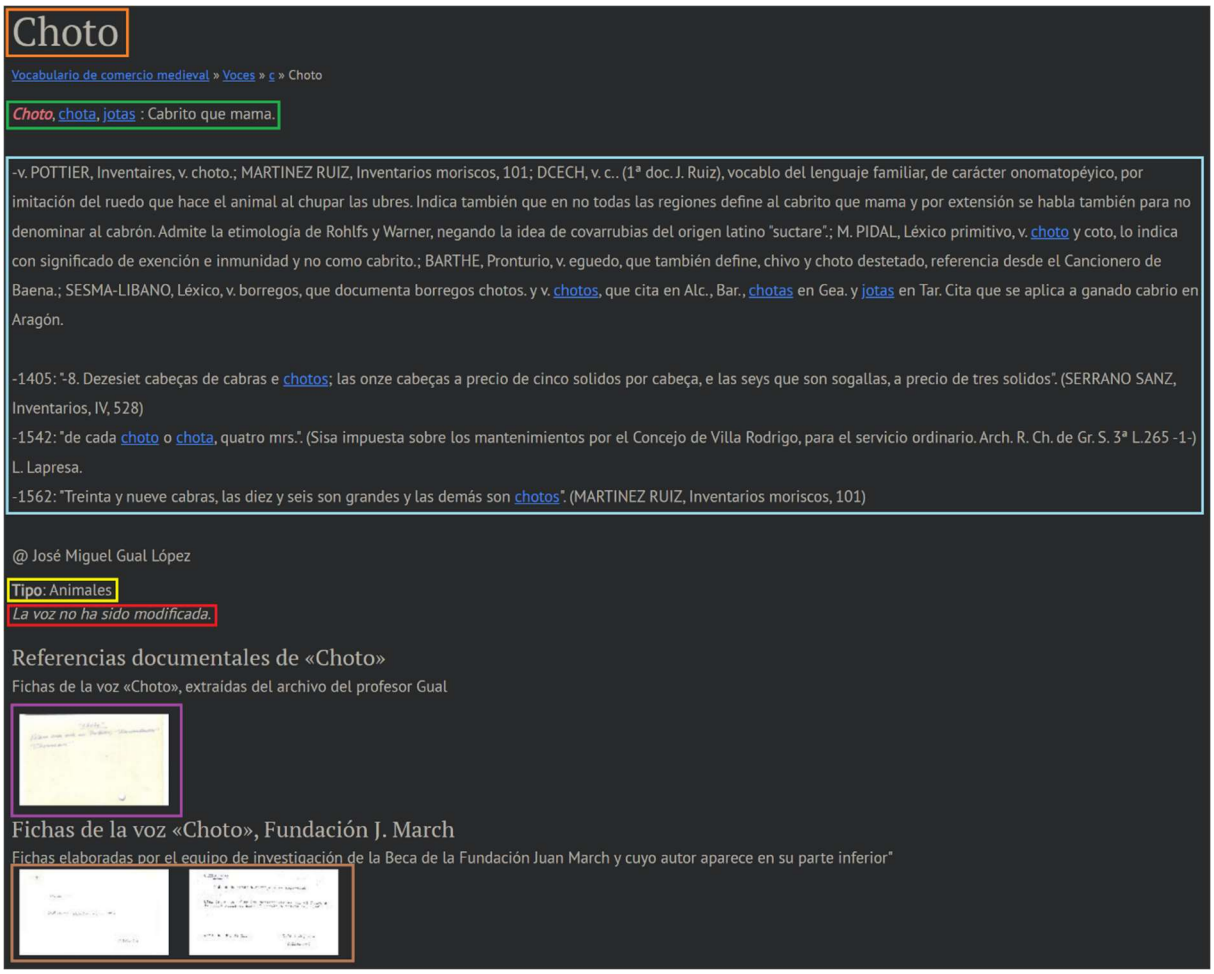
2.1 – Introduzione

In questo capitolo vengono riportate le metodologie e le strategie adottate per lo sviluppo del programma.

2.2 – Risoluzione del problema

Per poter estrarre i dati da una pagina web tramite la tecnica dello *scraping*[3], è necessario prima di tutto capire la struttura del sito e della pagina HTML per individuare gli elementi d'interesse che verranno poi usati come riferimenti dal software di *scraping*.
Si prenda ad esempio la pagina che tratta il vocabolo *Choto*:

[3] *Lo scraping è una forma di data mining che consiste nell'utilizzo di un software per estrapolare in maniera automatizzata dati da un determinato sito web*



[figura 2.1 – Schermata vocabolo *Choto*]

Come evidenziato in figura si possono distinguere:

- Il vocabolo (in arancione).
- Le varianti del vocabolo (in verde).
- La definizione (in azzurro).
- Il tipo (in giallo).
- La dicitura di aggiornamento (in rosso).
- I documenti referenziati dal vocabolo dall'archivio del professor Gual sotto forma di immagini (in viola).
- I documenti referenziati dal vocabolo dalla fondazione J. March sotto forma d'immagini (in marrone).

Ispezionando il codice HTML della pagina si può notare che quasi tutti questi elementi sono all'interno di un *tag* che ha come attributo o *id* o *class* (il che li rende univoci nella pagina HTML). Quelli che non hanno un identificatore *id* o *class* sono comunque legati ai precedenti e quindi facilmente recuperabili.

Infatti, se si ispezionasse il codice della pagina che è stata presa da esempio si vedrebbe che:

- Tutti i dati rilevanti sono nel *tag:*
  <*article class=”resultados”*>
  oppure
  <*article class=”static”*>.
- Il vocabolo è all’interno del *tag:*
  <*h3 class=”lexema_title”*>.
- Le varianti del vocabolo e la definizione sono contenuti nel *tag:*
  <*p class="descripcion"*>.
- Il tipo è all’interno del *tag*:
  <*span class="tipo"*>.
- La dicitura di aggiornamento non è in un *tag* con attributo ma la si può trovare subito dopo il tipo o dopo la descrizione all’interno di un *tag*:
  <*i*>.
- I documenti referenziati dall’archivio del professor Gual sono contenuti nel *tag*:
  <*div id="detalle_imagenes_lexema"*>.
- I documenti referenziati dall’archivio della fondazione J. March sono contenuti all’interno del *tag*:
  <*div id="imagenes_jmarch"*>.

Di conseguenza impostando lo *scraping* su questi *tag* univoci sarà possibile recuperare tutti i dati. La loro estrapolazione viene implementata nella funzione *scraping* che verrà approfondita nel paragrafo 4.5 “Funzione *scraping*”.

2.3 – Pianificazione e produzione del software

2.3.1 – Scelta del linguaggio di programmazione

Per la realizzazione di questa tesi è stato scelto il Python in quanto questo linguaggio di programmazione sta avendo una sempre più crescente diffusione e dispone anche di librerie specifiche utili per la risoluzione di questa tipologia di problema.

2.3.2 – Scelta delle librerie

Per poter svolgere lo *scraping* dei dati e collezionarli in un foglio elettronico sono state necessarie alcune librerie specifiche. Queste sono:

- *BeautifulSoup*: per la consultazione e la manipolazione del codice delle pagine HTML.
- *Requests*: per effettuare dei collegamenti a pagine Internet.
- *Openpyxl*: per la scrittura e la gestione dei file di estensione "*.xlsx"*.

La loro documentazione è disponibile agli indirizzi riportati nei punti 3 – 4 – 6 della "Bibliografia e sitografia".

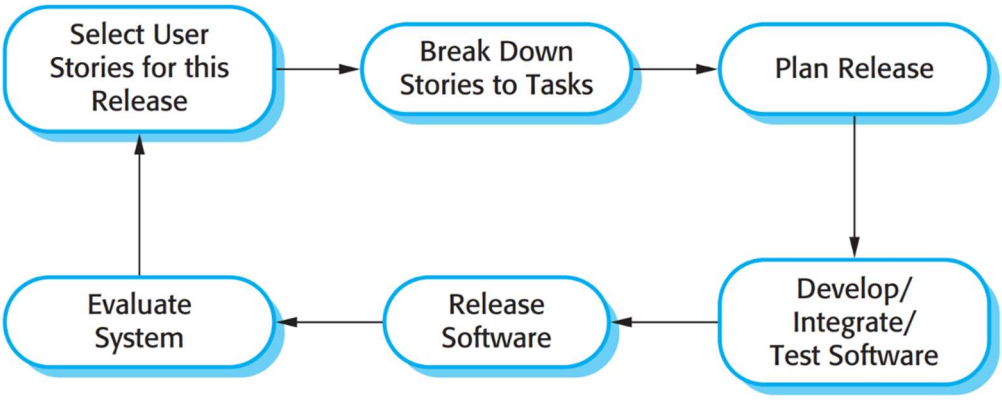
2.3.3 – Extreme programming (XP)

La tecnica di programmazione scelta per la realizzazione di questa tesi è l'*extreme programming* (abbreviata in *XP*). Sebbene le dimensioni di questo progetto non siano comparabili con i casi di utilizzo ideali (solitamente di media-grande dimensione con un discreto numero di programmatori che lavorano sullo stesso progetto), è stato comunque possibile utilizzare i suoi principi base.

Questa tecnica di produzione software predilige il rilascio di piccole e frequenti *release* dove vengono corretti *bug* e/o aggiunte nuove funzionalità alla *release* precedente. Inoltre, l'*XP* permette un maggiore coinvolgimento da parte del committente (in questo caso il relatore) nel suo sviluppo in quanto le modifiche al programma (definite *stories*) sono sempre concordate tra committente e sviluppatori. Infine, dopo ogni *release* vi è sempre un riscontro da parte del committente che permette di aver fin da subito un giudizio sulla qualità della produzione, andando ad evidenziare possibili criticità, modifiche necessarie o *bug* che non sono stati corretti in fase di produzione.

Per la realizzazione di questo programma sono stati rispettati due dei tre punti cardine dell'*extreme programming*:

- *Pair programming:* non è stato possibile farlo in quanto il programma è stato scritto da una sola persona.
- *Testing continuo:* durante lo sviluppo è sempre stato usato almeno un programma di testing (vedere paragrafo 2.5 "Testing del programma") per verificare il corretto funzionamento del programma.
- *Refactoring del codice:* durante il suo sviluppo si è cercato sempre di migliorare la qualità del codice senza modificarne il comportamento.



[figura 2.2 – ciclo di produzione *dell'Extreme programming*]

Ulteriori dettagli riguardanti l'*extreme programming* sono disponibili nel testo indicato nel punto 1 della "Bibliografia e sitografia".

2.4 – Distribuzione del software

Il software è stato distribuito tramite il portale *GitHub*. Il progetto è contenuto nella *repository* pubblica consultabile alla pagina "*https://github.com/FrancescoChemello/ProgrammaTesi*".
Al suo interno è possibile accedere alla sua documentazione (disponibile solamente per l'ultima versione), ad una guida sul suo utilizzo, ad un esempio completo del funzionamento del programma e alle varie *release* che sono state pubblicate durante lo sviluppo con le loro note di versione.

2.5 – Testing del programma

Per testare il comportamento del programma *WebScraping.py* è stato realizzato un altro programma denominato *CheckResults.py* che controlla che il database sia consistente, ovvero che non ci siano contraddizioni tra i dati salvati nel file di testo *links.txt* (contiene i dati che il programma ha trovato durante la fase di ricerca) con i dati che sono stati salvati nel foglio elettronico *data.xlsx* (contiene i dati che sono stati estrapolati).
Il programma *CheckResults.py* inizialmente recupera l'elenco dei link contenuti nel file *data.xlsx* e nel file *links.txt* e li salva in due liste denominate rispettivamente *records* e *links*. Successivamente, attraverso l'esecuzione di due cicli *for*, controlla che tutti gli elementi contenuti in *links* siano contenuti in *records* e viceversa. Se sono presenti elementi che rendono inconsistente il database, ovvero elementi che non sono presenti in entrambe le variabili, questi

vengono salvati su altre due liste per essere successivamente processati. La consistenza del database viene ristabilita attraverso le funzioni *add_link* (che aggiunte a *links.txt* gli elementi mancanti) e *remove_link* (che rimuove da *links.txt* gli elementi che sono in più). Infine, viene avvistato l'utente della presenza di una inconsistenza attraverso una stampa a video e viene generato un report che verrà salvato nel file *logerror.txt*.

Oltre a *CheckResults.py* sono stati creati anche altri programmi minori che però non verranno documentati in questa tesi.

Capitolo 3 – Funzionamento del programma

3.1 - Introduzione

In questo capitolo si vuole presentare il programma *WebScraping.py*, focalizzandosi sull'esecuzione e sulla struttura.

3.2 - Contenuto

Il file Python *WebScraping.py* effettua l'estrazione dei dati linguistici contenuti nel *Vocabulario del comercio medieval*.

Il programma recupera il collegamento ad ogni pagina che tratta un vocabolo, ne estrae il contenuto (nome, tipo, definizione, documenti referenziati, URL e la data di salvataggio) e li salva nel foglio elettronico che potrà poi essere usato come base per la realizzazione di un database terminologico. Inoltre, registra quali vocaboli sono stati salvati e quando, in modo tale da tenere il database sempre aggiornato e con tutte le voci presenti nel sito.

In aggiunta al file *WebScraping.py* è presente un altro programma, sempre in linguaggio Python, chiamato *CheckResults.py*, che serve a controllare che il database locale sia consistente con i dati registrati dal programma. Il suo funzionamento nel dettaglio viene presentato nel paragrafo 2.5 "Testing del programma".

Infine, sono presenti altre due cartelle: *data* e *result*, che contengono reciprocamente i file per l'esecuzione del programma e il foglio elettronico dove verranno salvati i dati estrapolati dal programma.

Più nello specifico nella cartella *data* sono presenti i seguenti file:

- *links.txt*: è un file di testo che contiene l'elenco dei vocaboli che sono stati estrapolati. I dati salvati sono gli indirizzi URL delle pagine che trattano i vocaboli e questi vengono salvati uno per riga.

- *lastrecord.txt*: è un file di testo che contiene il link dell'ultima voce che è stata salvata durante la fase di estrapolazione dei dati (presentata nel paragrafo 3.7 “Estrapolazione dati”). Questo file viene utilizzato per recuperare la sessione precedente qualora fosse stata interrotta; ciò avviene durante la fase di ripristino della precedente sessione (presentata nel paragrafo 3.5 “Ripristino della precedente sessione”).
- *lastsearch.txt:* è un file di testo che contiene il link dell'ultima pagina ispezionata durante la fase di ricerca (presentata nel paragrafo 3.6 “Ricerca”). Questo file viene utilizzato per recuperare la sessione precedente qualora fosse stata interrotta; ciò avviene durante la fase di ripristino della precedente sessione (presentata nel paragrafo 3.5 “Ripristino della precedente sessione”).
- *logerror.txt:* è un file che contiene il registro degli errori che sono avvenuti durante l'esecuzione del programma. I dati vengono scritti uno per riga riportando la data, l'interessato dall'errore e la tipologia d'errore.

Più nello specifico nella cartella *result* è presente il seguente file:

- *data.xlsx:* è un foglio elettronico che contiene tutti i dati estratti dal programma *WebScraping.py* sotto forma di tabella. I dati contenuti vengono salvati per riga e sono: vocabolo, definizione, documenti referenziati dall'archivio del professor Gual, documenti referenziati dalla fondazione J. March, numero di documenti referenziati, URL della pagina e data di salvataggio. Questi verranno approfonditi nel paragrafo 3.7 "Estrapolazione dati" e nel paragrafo 4.5 “Funzione *scraping*”.

3.3 - Struttura del Programma *WebScraping.py*

Il programma si può dividere in sei macro-funzioni che sono:

1. Modalità test.
2. Ripristino della precedente sessione.
3. Ricerca.
4. Estrapolazione dati.
5. Apertura link che hanno avuto un errore di connessione.
6. Aggiornamento dei vocaboli.

L'ordine in cui le macro-funzioni vengono presentate corrisponde anche all'ordine di esecuzione di queste ultime (ad eccezione della prima la quale verrà spiegata successivamente nel paragrafo 3.4 "Modalità test").

Ogni macro-funzione verrà analizzata successivamente in un paragrafo dedicato.

Inoltre, per il suo funzionamento sono state implementate dieci funzioni. Queste sono:

- Funzione *test.*
- Funzione *initialize.*
- Funzione *searching.*
- Funzione *scraping.*
- Funzione *table_write.*
- Funzione *update.*
- Funzione *swap.*
- Funzione *save.*
- Funzione *logerror.*
- Funzione *remove_link.*

Ogni funzione è stata documentata anche nel file Python secondo lo standard Epytext[4].

Le funzioni verranno presentate successivamente nel capitolo 4 "Le funzioni".

Infine, per la sua realizzazione sono state usate diverse librerie (alcune di queste già presentate nel paragrafo 2.3.2) che sono:

- *BeautifulSoup.*
- *Requests.*
- *Openpyxl.*
- *Datetime.*
- *Os.*
- *System.*
- *Msvcrt.*
- *Random.*

[4] *Lo standard Epytext è un linguaggio di markup per creare documentazione API strutturata. Vedere punto 8 della sezione "Bibliografia e sitografia".*

3.4 – Modalità di test

3.4.1 – Introduzione

La modalità di test è una fase opzionale che può esser avviata o meno da parte dall'utente. Quando viene lanciata, il programma è settato in modo che vengano eseguite tutte le sue macro-funzioni ma utilizzando un campione di vocaboli da processare ridotto.
Tale modalità si rileva particolarmente utile soprattutto durante le fasi dimostrative o di test.
Quando viene avviata, vengono cancellati i contenuti dei file: *links.txt, lastsearch.txt, lastrecord.txt* e *data.xlsx.*

3.4.2 – Funzionamento

Subito dopo l'avvio del programma viene chiesto all'utente se vuole utilizzare questa modalità. Se l'utente desiderasse provarla, gli basta digitare il tasto "y". Quindi si cancellano i file sopra elencati e il programma procede alla fase di ricerca (vedere paragrafo 3.6 "Ricerca"). Dopo aver invocato la funzione *initialize* (vedere paragrafo 4.3 "Funzione *initialize*"), viene chiamata la funzione *test* (vedere paragrafo 4.2 "Funzione *test*") che effettua un ridimensionamento della variabile *lett* a 15 elementi (la variabile *lett* contiene l'elenco delle pagine dove effettuare la ricerca – vedere paragrafo 4.2 "Funzione *initialize*"). Quindi il programma continuerà con le fasi elencate nel paragrafo 3.3 ma con un campione ridotto (solitamente di 300 elementi).

3.4.3 – Codice

```
x = input('Avviare il programma in modalità di test? (y/n)\nQuesto cancellerà
tutti i salvataggi precedenti!\n\n-> ')
#modalità di test
if x.casefold() == 'y':
    print('\nProgramma in modalità test')
    #calcello il file 'links.txt', 'lastsearch.txt' e 'lastrecord.txt' perché
non serve per la modalità test
    full_path = os.path.join(absolute_path, links_path)
    with open(full_path, 'r+', encoding='utf-8') as file:
        file.seek(0)
        file.truncate(0)
    full_path = os.path.join(absolute_path, lastsearch_path)
    with open(full_path, 'r+', encoding='utf-8') as file:
        file.seek(0)
        file.truncate(0)
    full_path = os.path.join(absolute_path, lastrecord_path)
    with open(full_path, 'r+', encoding='utf-8') as file:
        file.seek(0)
```

```
        file.truncate(0)
    #cancello anche il file 'data.xlsx' in quanto non serve per la modalità test
    full_path = os.path.join(absolute_path, data_path)
    if os.path.exists(full_path):
        os.remove(full_path)
    vartest = True

[. . .]

if vartest:
            lett = test(15, lett)
```

3.5 - Ripristino della precedente sessione

3.5.1 - Introduzione

Il ripristino della precedente sessione costituisce la prima parte del programma *WebScraping.py*. In questa fase vengono controllati i contenuti dei file *lasrecord.txt* e *lastsearch.txt* per verificare la presenza di una passata esecuzione che però non era stata completata[5]. Questo procedimento è stato introdotto per far sì che, in caso di interruzione del programma, potesse riprendere dal punto dov'era stato interrotto senza perdere il lavoro svolto precedentemente.

3.5.2 - Funzionamento

Per poter ripristinare la precedente sessione del programma, questo esegue (in ordine) i seguenti step:

1. Vengono estrapolati dal file *data.xlsx*, se presente, l'elenco dei link [6] dei vocaboli già presenti nel database. Questi vengono salvati nella variabile *old_link*.

2. Viene controllato il contenuto di *lastrecord.txt*. Se il file contiene un link e se l'utente desidera continuare la sessione precedente, il programma procede alla fase di estrapolazione dati e continua con il salvataggio dei record a partire dal successivo di quello indicato dal file. Quindi prosegue aprendo il file *links.txt*, cerca l'elemento

[5] *Il ripristino funziona solo in alcuni casi. Vedere paragrafi 3.6.2 – 3.7.2 – 3.8.2 per le casistiche ammesse. In caso di arresto improvviso o forzato attraverso CTRL + C non è garantito che il programma funzioni correttamente al successivo avvio e/o che i dati salvati siano leggibili.*

[6] *Sono contenuti nella colonna G "URL PAGINA" nel file data.xlsx. Vedere paragrafo 3.7.1.*

contenuto in *lastrecord.txt* e lo salva nella variabile *lastrecord*. Successivamente, copia nella variabile *link_risultati* i valori di *links.txt* dall'indice di *lastrecord* fino alla fine attraverso un ciclo *for*. Infine, salta direttamente alla fase di estrapolazione dei dati, illustrata nel paragrafo 3.7 "Estrapolazione dati".

Nel caso l'utente non volesse continuare, il programma cancella il contenuto del file *lastrecord.txt*, elimina tutti i link contenuti in *links.txt* che non sono stati salvati (questo per mantenere il database in uno stato di consistenza) e procede normalmente, seguendo tutti gli step elencati nel paragrafo 3.3 "Struttura del programma *WebScraping.py*".

3. Se *lastrecord.txt* fosse vuoto, allora il programma procede a controllare il contenuto di *lastsearch.txt*. Se dovesse contenere un link e se l'utente desidera continuare con la sessione precedente, allora il programma recupera i risultati della precedente ricerca ed imposta la nuova in modo tale da riprendere dal punto dove era stata interrotta. Quindi prosegue salvando nella variabile *lastsearch* il contenuto del file *lastsearch.txt* e salva i nuovi vocaboli che erano già stati trovati con la precedente ricerca in *link_risultati*. Dopo la chiamata alla funzione *initialize*, rimuove le pagine che sono già state analizzate attraverso l'esecuzione di un ciclo *for* che cancella gli elementi precedenti a *lastsearch* contenuti nella variabile *lett*.

   Nel caso l'utente non volesse continuare, il programma cancella il contenuto di *lastsearch.txt* ed elimina da *links.txt* tutti le voci che non sono state salvate (questo per mantenere il database in uno stato di consistenza). Infine, il programma procede normalmente, seguendo tutti gli step elencati nel paragrafo 3.3 "Struttura del programma *WebScraping.py*".

4. Se entrambi i file dovessero essere vuoti, il programma procede normalmente, seguendo tutte le fasi elencate nel paragrafo 3.3 "Struttura del programma *WebScraping.py*".

3.5.3 – Codice

```
#controllo precedenti salvataggi
    print('\nProgramma in modalità classica')
    #recupero l'elenco dei vocaboli che sono già stati salvati in 'data.xlsx'
    full_path = os.path.join(absolute_path, data_path)
    if os.path.exists(full_path):
        filexcel = openpyxl.load_workbook(full_path)
        excelread = filexcel.active
        for r in range(2, excelread.max_row+1):
```

```
            cell_obj = excelread.cell(row = r, column = 7)
            var = cell_obj.value
            old_link.append(str(var))
    full_path = os.path.join(absolute_path, lastrecord_path)
    with open(full_path, 'r+', encoding='utf-8') as file:
        if file.read() != '':
            x = input('Vuoi continuare il salvataggio dei record che era stato
interrotto l ultima volta? (y/n)\n\n-> ')
            if x.casefold() == 'y':
                #devo riportare il puntatore all'inizio file per poter rifare la
lettura
                file.seek(0)
                lastrecord = str(file.read())
                file.seek(0)
                file.truncate(0)
            else:
                #devo eliminare i link che non sono stati salvati per tener
consistente il database
                file.seek(0)
                temp = file.read()
                full_path = os.path.join(absolute_path, links_path)
                links = [r.rstrip('\n') for r in open(full_path, encoding='utf-
8')]
                index = links.index(temp)
                for i in range(index, len(links)):
                    remove_link(i)
                file.seek(0)
                file.truncate(0)
        else:
            full_path = os.path.join(absolute_path, lastsearch_path)
            with open (full_path, 'r+', encoding='utf-8') as file2:
                if file2.read() != '':
                    x = input('Vuoi continuare la ricerca che era stata
interrotta l ultima volta? (y/n)\n\n-> ')
                    if x.casefold() == 'y':
                        #continuo la precedente
                        #devo riportare il puntatore all'inizio file per poter
rifare la lettura
                        file2.seek(0)
                        lastsearch = file2.read()
                        file2.seek(0)
                        file2.truncate(0)
                        #aggiungo già ai link da estrapolare quelli trovati
nella precedente sessione
                        full_path = os.path.join(absolute_path, links_path)
                        link_risultati = [r.rstrip('\n') for r in
open(full_path, encoding='utf-8')]
                        for o in old_link:
                            if link_risultati.index(o) != -1:
```

```
                                link_risultati.pop(link_risultati.index(o))
                    else:
                        file2.seek(0)
                        file2.truncate(0)
                        #consistenza database
                        full_path = os.path.join(absolute_path, links_path)
                        link_risultati = [r.rstrip('\n') for r in
open(full_path, encoding='utf-8')]
                        for o in old_link:
                            if link_risultati.index(o) != -1:
                                link_risultati.pop(link_risultati.index(o))
                        for l in link_risultati:
                            remove_link(l)
                        link_risultati.clear()

[ . . . ]

print('Recupero vecchia sessione')
    full_path = os.path.join(absolute_path, links_path)
    #estrazione elementi da testo salvati in lista python con rimozione
carattere a capo
    old_link = [r.rstrip('\n') for r in open(full_path, encoding='utf-8')]
    index = old_link.index(lastrecord)
    for i in range(index+1, len(old_link)):
        link_risultati.append(old_link[i])
        cont_res = len(link_risultati)
```

3.6 - Ricerca

3.6.1 - Introduzione

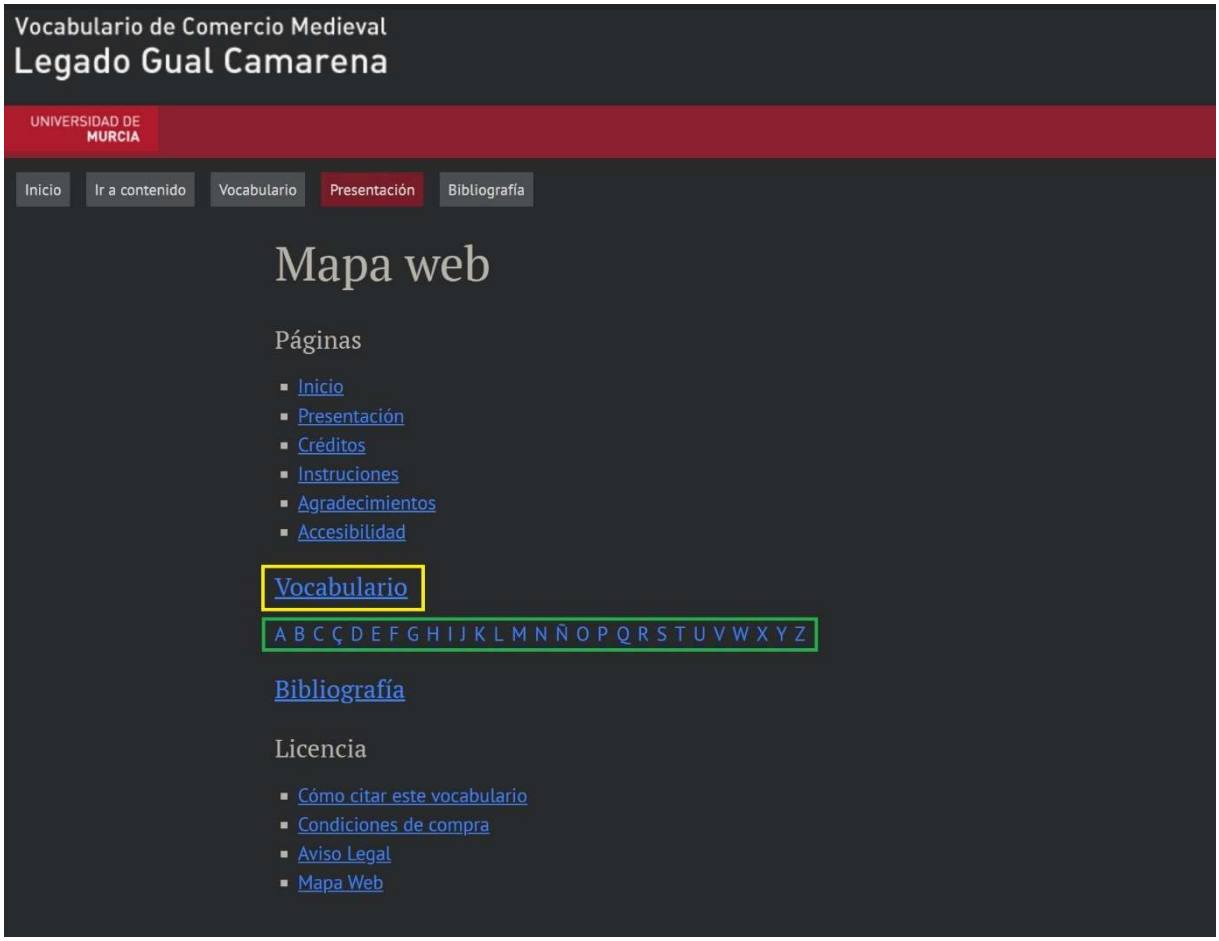
La fase di ricerca costituisce la seconda parte del programma *WebScraping.py* nella quale vengono recuperati tutti i link alle pagine che trattano vocaboli sul commercio medievale.

In questa sezione vengono utilizzate due funzioni: *initialize* (vedere paragrafo 4.3 “Funzione *initialize*”) e *searching* (vedere paragrafo 4.4 “Funzione *searching*”).

I link ai vocaboli vengono recuperati a partire dalla *Mapa web*, reperibile all’indirizzo *“https://www.um.es/lexico-comercio-medieval/index.php/mapaweb”*.

In questa pagina è presente la sezione *Vocabulario* (evidenziato in giallo in figura 3.1) dove, sotto di essa, è presente una serie di lettere che rappresentano la divisione per lettera iniziale dei vocaboli trattati dal sito (evidenziato in verde in figura 3.1). Durante questa fase, il

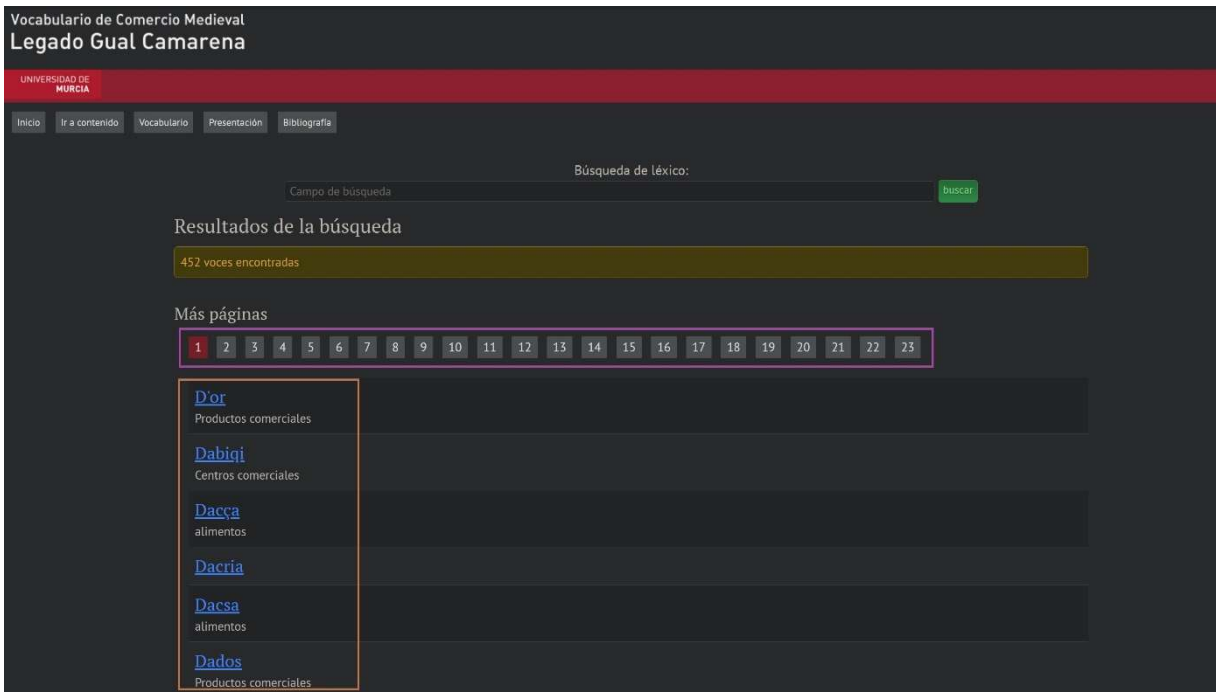
programma ispeziona ciascuna di esse e ne recupera l'elenco delle pagine (evidenziato in viola in figura 3.2).



[figura 3.1 – Mapa Web]

Infine, per ogni pagina vengono estrapolati tutti i link alle pagine che trattano un vocabolo (evidenziato in marrone in figura 3.2). Questi vengono salvati nella variabile *link_risultati* per essere successivamente elaborati.

Ogni link che viene aggiunto a *link_risultati* viene anche salvato nel file *links.txt*.



[figura 3.2 – Vocaboli divisi per lettera]

3.6.2 - Funzionamento

La funzione di ricerca comincia con la chiamata alla funzione *initialize* (vedere paragrafo 4.3 “Funzione *initialize*”) che recupera le pagine del vocabolario. I dati provenienti dalla funzione vengono salvati nella variabile lista *lett*.

Ora, per ogni elemento contenuto in *lett* viene invocata la funzione *searching* (vedere paragrafo 4.4 “Funzione *searching*”) che recupera i vocaboli presenti nella pagina e li salva nella variabile lista *res*. Infine, per ogni elemento in *res* viene controllato se è presente o in *old_link* o in *link_risultati* (rispettivamente la variabile che contiene la lista dei vocaboli già presenti nel database e la variabile che contiene la lista dei vocaboli che dovranno essere processati ed aggiunti al database). Se non fosse presente in nessuna delle due, allora il vocabolo verrebbe aggiunto al file *links.txt* tramite una scrittura ed aggiunto a *link_risultati*. Infine, viene salvata l'ultima ricerca fatta tramite la funzione *save* in modo da poter essere ripresa in caso di chiusura del programma (vedere paragrafo 3.5 "Ripristino della precedente sessione").

È possibile terminare anticipatamente la fase di ricerca inserendo da tastiera il carattere "q". L’interruzione viene gestita attraverso due *if* che controllano ad ogni iterazione del ciclo se è stato inserito o meno un carattere da tastiera e se questo corrisponde al carattere “q” (sia

maiuscolo che minuscolo). In caso di arresto anticipato, il programma salva in *lastsearch.txt* l'ultima ricerca effettuata e chiude il programma.

In caso di errore di risposta del sito[7], il programma salva l'ultima ricerca effettuata e termina la sua esecuzione. L'interruzione viene fatta in quanto se il server non rispondesse correttamente alle interrogazioni del programma, questo sarebbe impossibilitato a continuare con la sua corretta esecuzione.

Durante il processo viene stampato a video la percentuale di progressione che viene calcolata sulla dimensione della variabile *lett*.

3.6.3 – Codice

```
try:
        #inizializzazione
        print('Inizio ricerca --------------------------------')
        lett = initialize()
        if lastsearch != '':
            index = lett.index(lastsearch)
            temp = []
            for i in range(index+1, len(lett)):
                temp.append(lett[i])
            lett = temp
            del temp
        if vartest:
            lett = test(15, lett)
        tot = len(lett)
        cont = 1
        full_path = os.path.join(absolute_path, links_path)
        for r_link in lett:
            #genero nuovo link di ricerca
            sys.stdout.write('\r')
            percentage = (cont / tot)*100
            sys.stdout.write('Sto eseguendo la ricerca:
'+str(float(f'{percentage:.2f}'))+'%  premi q per interrompere -> ')
            sys.stdout.flush()
            res = searching(r_link)
            for l in res:
                if l not in old_link and l not in link_risultati:
                    #apertura file in modalità append per scrittura con raw
string come percorso
                    with open(full_path, 'a', encoding='utf-8') as f1:
                        #nuovo elemento da aggiungere al file di testo
```

[7] *In caso di una eccezione lanciata dal modulo requests il programma termina. Vedere punto 5 della "Bibliografia e sitografia" per le casistiche.*

```
                        f1.write(l)
                        f1.write('\n')
                    link_risultati.append(l)
                save(lastsearch_path, r_link)
            cont += 1
            if msvcrt.kbhit():
                if str(msvcrt.getwche()).casefold() == 'q':
                    sys.stdout.write('\nChiusura del programma ----------
-----------')
                    sys.stdout.flush()
                    sys.exit(1)
        del tot
        cont_res = len(link_risultati)
    #gestione degli errori:
    except requests.exceptions.RequestException as e:
        save(lastsearch_path, r_link)
        print('\n'+str(e)+'\nPer favore riprovare')
        sys.exit(1)
    print('\nRicerca completata ----------------------------')
    #cancello lastsearch in quanto ho finito con successo la ricerca
    full_path = os.path.join(absolute_path, lastsearch_path)
    with open(full_path, 'r+', encoding='utf-8') as file:
        file.seek(0)
        file.truncate(0)
    if len(link_risultati) != 0:
        print('Sono state trovate '+str(cont_res)+' nuove voci')
    else:
        print('Non sono state trovate nuove voci')
```

3.7 - Estrapolazione dati

3.7.1 - Introduzione

La fase di estrapolazione dati costituisce la terza parte del programma *WebScraping.py*. In questa parte vengono estratti tutti i dati linguistici dei vocaboli che sono stati recuperati durante la ricerca.

In questa sezione vengono utilizzate le funzioni *scraping* (vedere paragrafo 4.5 “Funzione *scraping*”) e *table_write* (vedere paragrafo 4.6 “Funzione *table_write*”).

I dati estratti per ogni vocabolo sono:

- Il vocabolo.

- La definizione.
- La tipologia.
- I documenti del professor Gual che sono stati referenziati dal vocabolo.
- I documenti della fondazione J. March che sono stati referenziati dal vocabolo.
- Il numero di documenti totali che sono stati referenziati.
- L'URL della pagina.
- La data di salvataggio del vocabolo nel database.

Questi vengono salvati nel foglio elettronico denominato *data.xlsx* contenuto nella cartella *result*.

3.7.2 - Funzionamento

La fase di estrapolazione dati comincia con il controllo della presenza del file *data.xlsx* nella cartella *result*. Se questo non fosse presente, il programma procederebbe a crearne uno nuovo e a caricarlo, altrimenti si limiterebbe soltanto a caricare quello esistente. Dopo aver recuperato l'indice di riga ed aver ridenominato il foglio di lavoro con la data corrente (così facendo è possibile sapere quando è stata la sua ultima esecuzione), il programma comincia ad estrarre i dati dei vocaboli contenuti nella variabile *link_risultati*. Quindi invoca prima la funzione *scraping* per estrarre i dati dalla pagina web e poi la funzione *table_write* per effettuarne la scrittura sul file *data.xlsx*. Infine, viene salvato il link appena esaminato sul file *lastrecord.txt* tramite la funzione *save* per poter ripristinare la sessione in caso di interruzione (vedere paragrafo 3.5 “Ripristino della precedente sessione”).

Al termine dell’estrazione di tutti i dati dei vocaboli contenuti in *link_risultati,* viene cancellato il contenuto di *lastrecord.txt*.

È possibile terminare anticipatamente la fase di estrapolazione dati inserendo da tastiera il carattere "q". L’interruzione viene gestita attraverso due *if* che controllano ad ogni iterazione del ciclo se è stato inserito un carattere da tastiera e se questo corrisponde al carattere “q” (sia maiuscolo che minuscolo). In caso di arresto anticipato, il programma salva il file *data.xlsx* e chiuderà il programma.

In questa fase possono essere catturate tre diverse eccezioni ed in base all'eccezione lanciata il programma si comporta in modo diverso. Nello specifico si ha:

- *requests.exceptions.ConnectTimeout*: in questo caso il collegamento tra il client (ovvero il programma) ed il server al sito non è stato eseguito in tempo e quindi la connessione viene chiusa. In tal caso il link interessato da questo errore viene salvato nella variabile *link_error* per essere processato successivamente (vedere paragrafo 3.8 "Apertura link che hanno avuto un errore di connessione"). Questo viene fatto in quanto il *ConnectTimeout* è un errore momentaneo che può dipendere dal carico della rete in quel momento. Riprovando in un tempo successivo è probabile che l'errore non si ripresenti.
- *BeautifulSoupException*: è un'eccezione che è stata costruita appositamente e viene lanciata quando la risposta del server alla connessione non è *HTTP 200 OK*. Questa eccezione può verificarsi ad esempio quando o il link non è più disponibile (HTTP 404 NOT FOUND) o che la risorsa non è più reperibile all'indirizzo indicato (HTTP 301 REDIRECT): in questo caso diventa impossibile continuare. Di conseguenza, il link interessato da questo errore viene rimosso tramite la funzione *remove_link* (vedere paragrafo 4.10 "Funzione *remove_link*") e l'errore viene registrato nel file di *log*.
  L'eccezione viene mostrata nel paragrafo 4.11 "*BeautifulSoupException"*.
- *requests.exceptions.RequestException*: in questo caso il collegamento al server non è andato a buon fine e quindi è impossibile continuare l'estrapolazione dei dati. In tal caso il link interessato da questo errore viene salvato nella variabile *link_error* per essere processato successivamente (vedere paragrafo 3.8 "Apertura link che hanno avuto un errore di connessione"). Questo viene fatto in quanto le eccezioni che cattura il *RequestException* (consultabili all'indirizzo riportato nel punto 5 della sezione "Bibliografia e sitografia") sono errori momentanei e possono dipendere dallo stato della rete in quel momento. Riprovando in un tempo successivo è probabile che l'errore non si ripresenti.

Durante il processo viene stampato a video la progressione che viene calcolata sulla dimensione della variabile *cont_res*.

3.7.3 – Codice

```
#estrapolazione dei dati dalle pagine html salvate in links.txt
it = 1
link_error = []
#apertura del file Excel
print('Inizio estrapolazione dati ---------------------')
full_path = os.path.join(absolute_path, data_path)
```

```
if not os.path.exists(full_path):
    excel_file = openpyxl.Workbook()
    worksheet = excel_file.active
    worksheet['A1'].value = 'VOCABOLO'
    worksheet['B1'].value = 'DEFINIZIONE'
    worksheet['C1'].value = 'TIPO'
    worksheet['D1'].value = 'DOCUMENTI REFERENZIATI DALL ARCHIVIO DEL PROFESSOR
GUAL'
    worksheet['E1'].value = 'DOCUMENTI REFERENZIATI DALLA FONDAZIONE J. MARCH'
    worksheet['F1'].value = 'NUMERO DOCUMENTI REFERENZIATI'
    worksheet['G1'].value = 'URL PAGINA'
    worksheet['H1'].value = 'DATA SALVATAGGIO'
    excel_file.save(absolute_path + '/result/data.xlsx')
    excel_file.close()
excel_file = openpyxl.load_workbook(full_path)
worksheet = excel_file.active
today = dt.datetime.now()
now = str(today.day) + '-' + str(today.month) + '-' + str(today.year)
#riga
cont = worksheet.max_row + 1
#così facendo il nome del foglio Excel indicherà sempre la data dell'ultimo
aggiornamento
worksheet.title = now
del now
for l in link_risultati:
    try:
        sys.stdout.write('\r')
        sys.stdout.write('Estrapolazione dati:
'+'['+str(it)+'/'+str(cont_res)+']  premi q per interrompere -> ')
        sys.stdout.flush()
        res = scraping(l)
        table_write(res, worksheet, cont)
        cont += 1
        it += 1
        save(lastrecord_path, l)
        if msvcrt.kbhit():
            if str(msvcrt.getwche()).casefold() == 'q':
                sys.stdout.write('\nSalvataggio file data.xlsx ----------------
----')
                sys.stdout.flush()
                excel_file.save(full_path)
                excel_file.close()
                sys.stdout.write('\r')
                sys.stdout.write('Chiusura del programma ----------------------
-')
                sys.stdout.flush()
                sys.exit(1)
    except requests.exceptions.ConnectTimeout:
        link_error.append(l)
        #eseguo lo swap
        swap(l)
        cont_res -= 1
```

```
        continue
    except BeautifulSoupException as e:
        logerror(l, e)
        cont_res -= 1
        #rimuovo il link non valido
        remove_link(l)
        continue
    except requests.exceptions.RequestException as e:
        link_error.append(l)
        #eseguo lo swap
        swap(l)
        cont_res -= 1
        continue
excel_file.save(full_path)
full_path = os.path.join(absolute_path, lastrecord_path)
with open(full_path, 'r+', encoding='utf-8') as file:
    file.seek(0)
    file.truncate(0)
```

3.8 - Apertura link che hanno avuto un errore di connessione

3.8.1 - Introduzione

L'apertura dei link che hanno avuto un errore di connessione costituisce la terza parte del programma *WebScraping.py*. Questa non sempre viene eseguita in quanto dipende dalla buona riuscita o meno della fase precedente.

Qui viene effettuata nuovamente l'estrapolazione dei dati per i link che avevano avuto un problema di connessione (vedere elenco puntato nel paragrafo 3.7.2) con modalità simili a quelle descritte nella fase di estrapolazione dati.

3.8.2 - Funzionamento

Inizialmente vengono rimossi da *links.txt* tutti gli elementi contenuti nella variabile *link_error* (questo perché in caso di arresto il programma mantenga comunque lo stato di coerenza). Poi si procede al recupero dei dati linguistici che, come funzionamento, è analogo a quello già descritto nella fase di estrapolazione dati (vedere paragrafo 3.7 “Estrapolazione dati”) ad eccezione del fatto che non viene più registrato in *lastrecord.txt* l’ultimo link analizzato.

Gli elementi contenuti in *link_error* vengono processati attraverso un ciclo *while* che ispeziona sempre l'elemento in testa alla lista. Se si dovesse ripresentare un *ConnectTimeout* o un *RequestException*, il programma sposterebbe in coda a *link_error* il link interessato per poterlo riprocessare successivamente. Se l'estrapolazione dei dati dovesse andare a buon fine, i dati estratti verrebbero salvati e il link verrebbe aggiunto a *links.txt*. Infine, il link appena salvato verrebbe rimosso da *link_error*.

Il numero di errori ammessi è però limitato dal valore della variabile intera *maxfail* (che è uguale alla lunghezza di *link_error* più la parte intera della sua metà).

Il ciclo continua finché o tutti i link in *link_error* sono stati estrapolati correttamente (ovvero fino a quando *link_error* sarà vuota) o il numero di tentativi falliti ha superato quello di *maxfail*. Al termine, gli eventuali link ancora presenti in *link_error* vengono persi.

Durante il processo viene stampato a video la progressione che viene calcolata sulla dimensione della variabile *c*.

3.8.3 – Codice

```
#gestione pagine non caricate
for l in link_error:
    remove_link(l)
cont_res = len(link_error)
c = 1
if len(link_error) > 0:
    fail = 0
    maxfail = len(link_error) + (len(link_error) // 2)
    while fail < maxfail and len(link_error) > 0:
        #ripeto il codice
        l = link_error[0]
        try:
            #rifaccio l'analisi
            sys.stdout.write('\r')
            sys.stdout.write('Estrapolazione dati voci con errore
connessione'+'['+str(c)+'/'+str(cont_res)+']'+'premi q per interrompere -> ')
            sys.stdout.flush()
            res = scraping(l)
            cont += 1
            table_write(res, worksheet, cont)
            link_error.remove(l)
            #è stato salvato, lo aggiungo nuovamente
 all'elenco dei link
            with open(os.path.join(absolute_path, links_path), 'w',
encoding='utf-8') as file:
                file.write(l)
```

```
            if msvcrt.kbhit():
                if str(msvcrt.getwche()).casefold() == 'q':
                    sys.stdout.write('\nSalvataggio file data.xlsx ------------
--------')
                    sys.stdout.flush()
                    excel_file.save(full_path)
                    excel_file.close()
                    sys.stdout.write('\r')
                    sys.stdout.write('Chiusura del programma ------------------
-----')
                    sys.stdout.flush()
                    sys.exit(1)
        except requests.exceptions.ConnectTimeout:
            logerror(l, 'requests.exceptions.ConnectTimeout')
            #riprovo successivamente a ricollegarmi alla pagina
            link_error.remove(l)
            link_error.append(l)
            continue
        except BeautifulSoupException as e:
            logerror(l, e)
            link_error.remove(l)
            continue
        except requests.exceptions.RequestException as e:
            logerror(l, e)
            cont_res -= 1
            link_error.remove(l)
            continue
    excel_file.save(full_path)
#rimozione record che non è stato possibile gestire per errore connessione
del link_error, cont
#chiusura file Excel
excel_file.close()
print('\nEstrapolazione dati completata ----------------')
```

3.9 - Aggiornamento dei vocaboli

3.9.1 - Introduzione

L'aggiornamento dei vocaboli costituisce l'ultima parte del programma *WebScraping.py*. In questa fase viene controllata l'eventuale presenza di un aggiornamento per tutti i vocaboli contenuti nel foglio elettronico *data.xlsx*.

Per poter effettuare l'aggiornamento si sfrutta il fatto che se una voce venisse modificata, la sua relativa pagina riporterebbe la dicitura "*La voz ha sido modificada a fecha ANNO-MESE-*

GIORNO". Confrontando la data di salvataggio con quella riportata nel sito è possibile determinare se una voce è stata modificata e se quindi necessiti di un aggiornamento o meno.

3.9.2 - Funzionamento

Inizialmente vengono recuperati dal file *data.xlsx* la coppia URL e data di salvataggio di ogni vocabolo. Questi vengono salvati nella tupla denominata *tupla* ed aggiunti alla variabile lista *links*. Poi, per ogni elemento di *links*, viene invocata la funzione *update* (vedere paragrafo 4.7 "Funzione *update*") per l'aggiornamento dei record ad eccezione di quelli appena salvati. Al termine viene salvato il file *data.xlsx* ed il programma termina.

In questa fase viene controllata anche la presenza di righe vuote all'interno del file *data.xlsx* che, se presenti, vengono segnalate all'utente tramite il file *logerror.txt*.

È possibile terminare anticipatamente la fase di aggiornamento inserendo da tastiera il carattere "q". L'interruzione viene gestita attraverso due *if* che controllano ad ogni iterazione del ciclo se è stato inserito un carattere da tastiera e se questo corrisponde al carattere "q" (sia maiuscolo che minuscolo). In caso di arresto anticipato, il programma salva il file *data.xlsx* e chiude il programma.

Durante il processo viene stampato a video la percentuale di progressione che viene calcolata sulla lunghezza della variabile *links*.

3.9.3 – Codice

```
print('Aggiornamento vocaboli ------------------------')
full_path = os.path.join(absolute_path, data_path)
excel_file = openpyxl.load_workbook(full_path)
worksheet = excel_file.active
links = []
#recupero link dal file Excel
for r in range(2, worksheet.max_row+1):
    sys.stdout.write('\r')
    percentage = (r / worksheet.max_row)*100
    sys.stdout.write('Inizializzazione: '+str(float(f'{percentage:.2f}'))+'%  ')
    sys.stdout.flush()
    #recupero link
    cell_obj = worksheet.cell(row = r, column = 7)
    var1 = str(cell_obj.value)
    if var1 == 'None':
        error = 'riga '+str(r)+' vuota!'
        #scrivo che è presente un errore
```

```
            logerror('data.xlsx', error)
            tupla = (var1, var1)
            links.append(tupla)
            continue
        #recupero data
        cell_obj = worksheet.cell(row = r, column = 8)
        var2 = str(cell_obj.value)
        rgg = int(var2[0:var2.find('-')])
        rmm = int(var2[var2.find('-')+1:var2.rfind('-')])
        raa = int(var2[var2.rfind('-')+1: len(var2)])
        record_date = dt.datetime(raa, rmm, rgg)
        tupla = (var1, record_date)
        links.append(tupla)
cont = 2
for l in links:
    try:
        sys.stdout.write('\r')
        percentage = ((cont-1) / len(links))*100
        sys.stdout.write('Aggiornamento in corso:
'+str(float(f'{percentage:.2f}'))+'%  '+'premi q per interrompere -> ')
        sys.stdout.flush()
        #se c'è un eventuale spazio nel database devo comunque saltarlo
        if l[0] != 'None':
            #evito di rifare il controllo dell'aggiornamento per i vocaboli
appena inseriti
            if str(today.date()) != record_date:
                #procedo all'aggiornamento
                update(l[1], l[0], worksheet, cont)
        cont += 1
        if msvcrt.kbhit():
                if str(msvcrt.getwche()).casefold() == 'q':
                    sys.stdout.write('\nSalvataggio file data.xlsx ----------
--------')
                    sys.stdout.flush()
                    excel_file.save(full_path)
                    excel_file.close()
                    sys.stdout.write('\r')
                    sys.stdout.write('Chiusura del programma ----------------
-----')
                    sys.stdout.flush()
                    sys.exit(1)
    #se ho un errore di connessione non importa, si riproverà la prossima volta
    #non salvo niente in logerror
    except requests.exceptions.ConnectTimeout:
        cont += 1
        continue
    except BeautifulSoupException as e:
        cont += 1
        continue
    except requests.exceptions.RequestException as e:
        cont += 1
        continue
```

```
excel_file.save(full_path)
excel_file.close()
print('\nAggiornamento completato ----------------------')
```

Capitolo 4 - Le funzioni

4.1 - Introduzione

In questo capitolo verranno analizzate le funzioni che sono state implementate nel programma *WebScraping.py*. Verranno evidenziate la struttura, il funzionamento e i metodi utilizzati.

4.2 – Funzione *test*

4.2.1 – Introduzione

Questa funzione effettua il ridimensionamento della variabile *lett* (vedere paragrafo 4.3 "Funzione *initialize*") che contiene l'elenco delle pagine del vocabolario dove effettuare la ricerca dei vocaboli.

4.2.2 – Funzionamento

La funzione, attraverso un ciclo *while*, elimina randomicamente un elemento dalla variabile *lett*. Il ciclo continua finché la lunghezza di *lett* non è uguale a *num* (che nel programma è stato messo a 15). Al termine viene restituita *lett* al chiamante.

4.2.3 – Codice

```
def test(num, lett):
    """
    Funzione di test del programma.

    parametri:
        @type num: int
        @param num: parametro di ridimensionamento di lett
        @type lett: list
        @param lett: lista che contiene l'elenco delle pagine trovate dalla
funzione initialize
    return:
        @rtype: list
        @return: lista che contiene l'elenco delle pagine trovate dalla funzione
initialize con resize a num
    """
    while(len(lett) > num):
        index = random.randint(0, len(lett)-1)
```

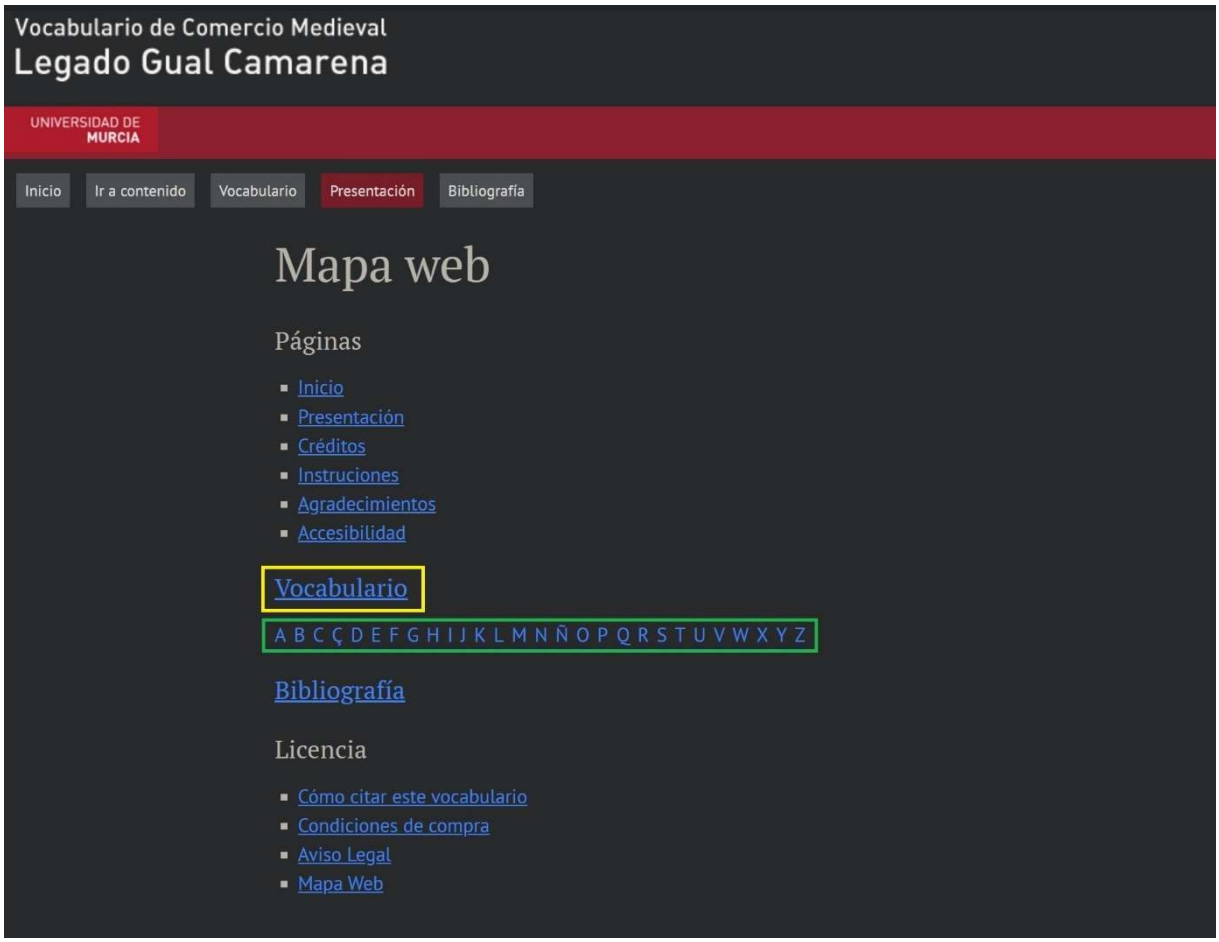
```
        lett.pop(index)
    return lett
```

4.3 – Funzione *initialize*

4.3.1 – Introduzione

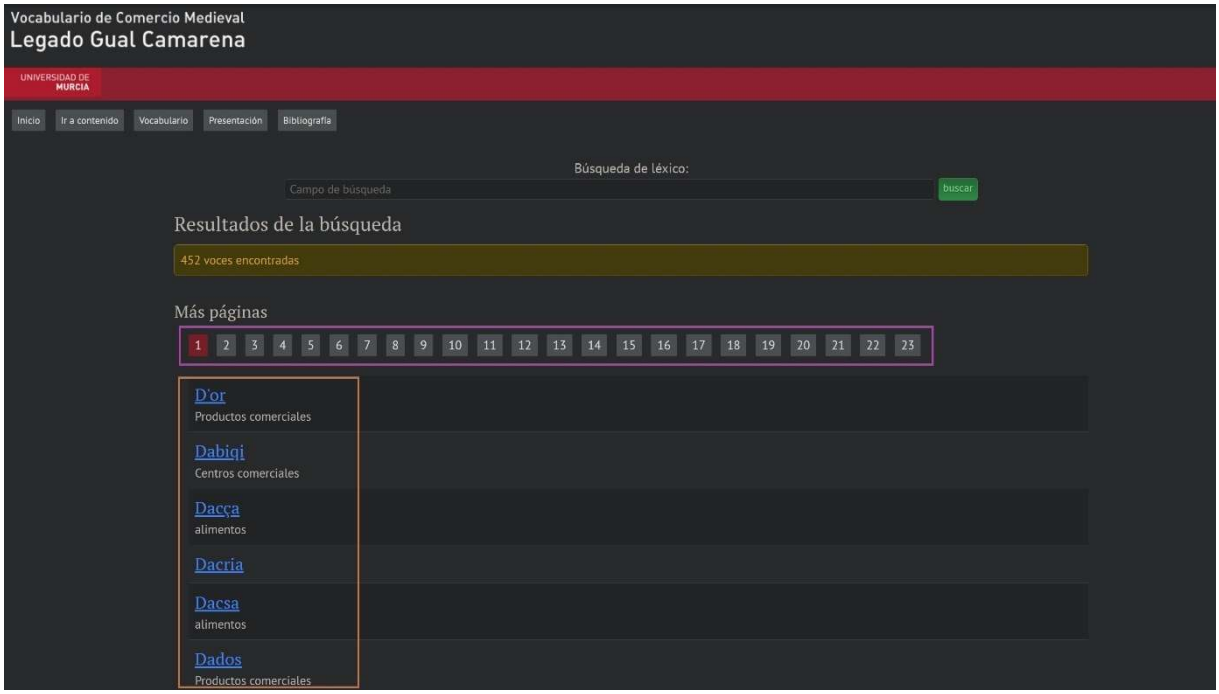
La funzione *initialize* serve per recuperare l'elenco delle pagine dove i vocaboli vengono suddivisi.

Alla pagina *"https://www.um.es/lexico-comercio-medieval/index.php/mapaweb"*, sotto la voce *Vocabulario* (in figura 4.1 in giallo), sono presenti una serie di lettere che fungono da filtri per lettera iniziale dei vocaboli presenti nel sito (in figura 4.1 in verde).



[Figura 4.1 – Mapa web]

Al loro interno, i vocaboli che iniziano per una determinata lettera vengono suddivisi a gruppi di 20 vocaboli per pagina e quest'ultime vengono riportate sotto forma di elenco nella parte alta del sito sotto la voce *Más páginas* (in figura 4.2 in viola).



[figura 4.2 – Vocaboli divisi per lettera]

La funzione ne recupera l'elenco e li rimanda, sotto forma di lista, al chiamante.

4.3.2 – Funzionamento

La funzione inizialmente si collega alla pagina "*https://www.um.es/lexico-comercio-medieval/index.php/mapaweb*" e seleziona il *tag <ul class="diccionario">* che contiene i filtri per lettera iniziale. Dopodiché recupera tutti i *tag href* (ovvero tutti i collegamenti ipertestuali) contenuti nella sezione evitando però di copiarne i doppioni.
Successivamente, per ogni link trovato, si collega alla pagina corrispondente e ne estrapola tutti i *tag href* delle pagine di navigazione. Queste vengono salvate, sempre evitando di copiarne i doppioni, su una variabile denominata *res* che poi viene restituita al chiamante.

4.3.3 – Codice

```
def initialize():
    """
    Funzione che serve per recuperare l'elenco di tutte le pagine del
vocabolario mappate dalla Mapa Web.
```

```
    parametri:
        None
    return:
        @rtype: list
        @return: lista che contiene l'elenco delle pagine trovate
    """
    res = []
    lett = []
    #link mapsite
    mapsite = 'https://www.um.es/lexico-comercio-medieval/index.php/mapaweb'
    #mi collego alla pagina
    response = requests.get(mapsite)
    response.raise_for_status()
    #avvio scraping
    soup = bs4.BeautifulSoup(response.content, 'html.parser')
    #selezione risultati
    risultati = soup.find('ul', class_='diccionario')
    a_risultati = risultati.find_all('a')
    #estrapolazione href
    for i in a_risultati:
        vrf = str(i.get('href'))
        if vrf not in res:
            #salvo la pagina
            lett.append(vrf)
    #ora devo ottenere tutte le pagine della lettera
    del response, soup, risultati, a_risultati
    cont = 1
    for l in lett:
        sys.stdout.write('\r')
        percentage = (cont / len(lett))*100
        sys.stdout.write('Inizializzazione:
'+str(float(f'{percentage:.2f}'))+'%  ')
        sys.stdout.flush()
        #connessione alla pagina della lettera
        response = requests.get(l)
        response.raise_for_status()
        #avvio scraping
        soup = bs4.BeautifulSoup(response.content, 'html.parser')
        #selezione risultati
        risultati = soup.find('nav', class_='pagination')
        if risultati is not None:
            a_risultati = risultati.find_all('a')
            for i in a_risultati:
                vrf = str(i.get('href'))
                if vrf not in res:
                    res.append(vrf)
        cont += 1
    return res
```

4.4 – Funzione *searching*

4.4.1 – Introduzione

La funzione *searching* recupera tutti i link ai vocaboli contenuti nelle pagine recuperate attraverso la funzione *initialize* (vedere paragrafo 4.3 “Funzione *initialize*”).

4.4.2 – Funzionamento

La funzione si collega alla pagina indicata dal parametro di tipo stringa s*h_link* (questo parametro conterrà un elemento trovato dalla funzione *initialize*).
Dopo aver ottenuto il contenuto della pagina, la funzione ricerca l’elenco dei vocaboli nella porzione indicata dal *tag* <*article class="resultados"*>. Quindi seleziona il secondo *tag ul* e ne estrae tutti i collegamenti ipertestuali che li salva, evitando di copiarne i doppioni, in una variabile denominata *res* che viene poi restituita al chiamante.

4.4.3 – Codice

```
def searching(sh_link):
    """
    Funzione che ricerca i vocaboli presenti in una determinata pagina.

    parametri:
        @type sh_link: str
        @param sh_link: contiene il link della pagina dove viene eseguita la
ricerca
    return:
        @rtype: list
        @return: lista contenente tutti i link a pagine di vocaboli
    """
    res = []
    #http request -get http
    response = requests.get(sh_link)
    response.raise_for_status()
    #estraggo il testo della risposta in formato html
    soup = bs4.BeautifulSoup(response.content, 'html.parser')
    #sezione risultati
    soup = soup.find('article', class_='resultados')
    skip = soup.find_next('ul', id=None)     #salto il primo <ul>
    risultati = skip.find_next('ul', id=None)
    #trovo tutti gli hyperlink della sezione risultati
    a_risultati = risultati.find_all('a')
    #gestione link ottenuti
    for i in a_risultati:
        vrf = str(i.get('href'))
```

```
        if vrf not in res:
            res.append(vrf)
    return res
```

4.5 – Funzione *scraping*

4.5.1 – Introduzione

La funzione *scraping* serve per estrapolare i dati linguistici delle pagine web relative ai vocaboli (vedere paragrafo 3.7 “Estrapolazione dati” e paragrafo 2.2. “Risoluzione del problema”).

4.5.2 – Funzionamento

La funzione inizialmente si collega alla pagina indicata dal parametro *sr_link* bloccandone i reindirizzamenti. Poi viene controllato che lo *status code* della risposta del server sia 200 (*HTTP 200 OK*) e che quindi il collegamento al server sia stato eseguito correttamente. In caso di *status code* diverso viene lanciata l’eccezione *BeautifulSoupException* (vedere paragrafo 4.11 “*BeautifulSoupException*”).

Dopo aver ottenuto il contenuto della pagina, la funzione estrapola il primo blocco (vedere punto 1 paragrafo 2.2 “Risoluzione del problema”). Poi procede estraendo il vocabolo, la definizione e il tipo. In seguito, estrapola i documenti referenziati tenendo conto del numero di *tag alt* che processa attraverso la variabile intera *num_doc*. Infine, salva l’URL della pagina (che è il parametro *sr_link*) e la data corrente (sfruttando la libreria *datetime*) in formato dd/mm/yy.

Per evitare di eseguire un’operazione di estrapolazione su un *tag* inesistente (in quanto non tutte le pagine presentano per forza tutti i parametri richiesti) viene controllato sempre che il risultato della selezione del *tag* sia diversa da *None*[8]. In caso contrario quel dato viene ignorato e verrà salvata una stringa vuota al suo posto.

Tutti i dati vengono salvati nella variabile denominata *res*, di tipo lista, che viene poi restituita al chiamante.

[8] *Il metodo find della libreria BeautifulSoup quando non trova il tag richiesto all’interno della pagina HTML restituisce “None” come valore.*

4.5.3 – Codice

```
def scraping(sr_link):
    """
    Funzione che esegue lo scraping di dati della pagina web. Vengono estratti:
        -VOCABOLO: la parola interessata
        -DEFINIZIONE: la descrizione del termine
        -TIPO: la tipologia del termine
        -DOCUMENTI REFERENZIATI: contiene tutti i documenti che sono stati
referenziati

    parametri:
        @type sr_link: str
        @param sr_link: link della pagina da analizzare
    return:
        @rtype: list
        @return: lista contenente i dati estratti
    """
    res = []
    #lettura pagina html
    #blocco reindirizzamento a siti esterni
    response = requests.get(sr_link, allow_redirects=False)
    #per il lancio delle eccezioni HTTP durante la creazione
    response.raise_for_status()
    if response.status_code != 200:
        raise BeautifulSoupException(sr_link, response.status_code)
    page = bs4.BeautifulSoup(response.content, 'html.parser')
    #selezione contenuto della pagina
    soup = page.find('article', class_='resultados')
    if soup is None:
        #esiste anche questa variazione
        soup = page.find('article', class_='static')
    #estrapolo il vocabolo
    vocabolo = ''
    v = soup.find('h3', class_='lexema_title')
    if v is not None:
        vocabolo = v.text.strip()
    res.append(vocabolo)
    #estraggo la definizione
    definizione = ''
    df = soup.find('p',  class_='descripcion')
    if df is not None:
        definizione = df.text.strip()
        if str('\n') in definizione:
            definizione.replace('\n', ' ')
    res.append(definizione)
    #estraggo il tipo
    tipo = ''
    t = soup.find('span', class_='tipo')
    if t is not None:
        tipo = t.text.strip()
    res.append(tipo)
```

```
    #estraggo archivio documenti referenziati
    num_doc = 0
    arch_def = ''
    arch = soup.find('div', id='imagenes')
    if arch is not None:
        a_risultati = arch.find_all('a', alt=True)
        #gestione alt tags ottenuti
        for i in a_risultati:
            vrf = str(i.get('alt'))
            arch_def +=str(vrf) + '; '
            num_doc += 1
    res.append(arch_def)
    #estraggo archivio documenti referenziati
    arch_def = ''
    arch = soup.find('div', id='imagenes_jmarch')
    if arch is not None:
        a_risultati = arch.find_all('a', alt=True)
        #gestione alt tags ottenuti
        for i in a_risultati:
            vrf = str(i.get('alt'))
            arch_def +=str(vrf) + '; '
            num_doc += 1
    res.append(arch_def)
    res.append(num_doc)
    res.append(sr_link)
    date = dt.datetime.now()
    now = str(date.day) + '-' + str(date.month) + '-' + str(date.year)
    res.append(now)
    return res
```

4.6 – Funzione *table_write*

4.6.1 – Introduzione

La funzione *table_write* serve per trascrivere i dati estrapolati dalla funzione *scraping* (vedi paragrafo 4.5 “Funzione *scraping*”) nel foglio elettronico *data.xlsx*.

4.6.2 – Funzionamento

La funzione, attraverso un ciclo *for*, trascrive ogni elemento contenuto in *res* nel foglio *data.xlsx*. La posizione della scrittura viene determinata dalla coppia *col* (una variabile interna alla funzione inizializzata ad 1) e *r* (variabile passata come argomento).
Ad ogni iterazione del ciclo la variabile *col* viene incrementata di 1 permettendo così di scrivere i dati in riga.

4.6.3 – Codice

```
def table_write(res, worksheet, r):
    """
    Funzione che esegue la scrittura dei dati estrapolati nel file Excel. I file
vengono scritti per righe.

    parametri:
        @type res: list
        @param res: contiene i dati estratti dalla funzione scraping
        @type worksheet: openpyxl
        @param worksheet: contiene la pagina di lavoro Excel nella quale
verranno scritti i record
        @type r: int
        @param r: contatore della riga di scrittura
    return:
        None
    """
    col = 1
    for item in res:
        #scrittura per riga
        w = worksheet.cell(row = r, column = col)
        w.value = item
        col += 1
```

4.7 – Funzione *update*

4.7.1 – Introduzione

La funzione *update* serve per verificare che i dati salvati nel foglio elettronico *data.xlsx* siano uguali a quelli presenti nel sito. Come riportato nel paragrafo 2.2 “Risoluzione del problema”, è presente la dicitura che riporta se il vocabolo è stato modificato o meno e se sì in quale data. Confrontando questa data con la data di salvataggio del record, la funzione determina se un vocabolo debba esser aggiornato.

4.7.2 – Funzionamento

La funzione inizialmente si collega alla pagina indicata dal parametro *up_link*. Successivamente ne recupera il contenuto e ne estrapola il primo blocco (vedere paragrafo 2.2. “Risoluzione del problema”). Avendo contenuto differente, la dicitura di modifica del vocabolo non sempre è reperibile nella stessa posizione all’interno della pagina. Di conseguenza la funzione controlla in due diverse posizioni:

- il primo *tag* <*i*> contenuto nella sezione <*span class=”tipo”*>.
- Il primo *tag* <*i*> dopo il secondo *tag* <*p*> contenuto nella sezione <*p class=”descripcion”*>.

Una volta ottenuto il contenuto della stringa procede al controllo della data. Qui viene confrontata la data estrapolata (divisa in giorno, mese ed anno e ricostruita come oggetto di classe *datetime* chiamata *dtonline*) con la data passata come argomento denominata *rdate*. Se la data indicata da *rdate* è antecedente a quella di *dtonline*, allora si procede con una nuova estrapolazione dei dati richiamando la funzione *scraping* (vedere paragrafo 4.5 “Funzione *scraping*”). Infine, sempre in caso di necessario aggiornamento, si richiama la funzione *table_write* (vedere paragrafo 4.6 “Funzione *table_write*”) che ne sovrascriverà i dati. La sovrascrittura dei dati viene fatta grazie alla variabile *row* che contiene il numero di riga dov’è salvato il vocabolo nel foglio elettronico.

4.7.3 – Codice

```
def update(rdate, up_link, worksheet, row):
    """
    Funzione che esegue l'aggiornamento dei record nel database.

    parametri:
        @type: rdate: datetime
        @param rdate: data di salvataggio del record nel database
        @type up_link: str
        @ param up_link: link della pagina
        @type: worksheet: openpyxl
        @param worksheet: contiene la pagina di lavoro Excel nella quale sono
scritti i record
        @type row: int
        @param row: riga dove il vocabolo è stato salvato
    return:
        None
    """
    res = []
    #connessione alla pagina
    response = requests.get(up_link)
    response.raise_for_status()
    #scraping data
    page = bs4.BeautifulSoup(response.content, 'html.parser')
    soup = page.find('article', class_='resultados')
    if soup is None:
        #esiste anche questa variazione
        soup = page.find('article', class_='static')
    #prima cerco per tipo
    d = None
    t = soup.find('span', class_='tipo')
```

```
    if t is not None:
        #continuo dopo il tipo
        d = t.find_next('i')
    else:
        #cerco dopo la descrizione
        t = soup.find('p', class_='descripcion')
        if t is not None:
            p = t.find_next('p', id=None)
            d = p.find_next('i')
    #può anche non esserci
    if d is not None:
        date = d.text.strip()
        #viene scritto se la voce non è stata modificata
        if date.find('no') == -1:
            #scompongo date in giorno, mese ed anno
            daa = int(date[date.find('-')-4:date.find('-')])
            dmm = int(date[date.find('-')+1:date.rfind('-')])
            dgg = int(date[date.rfind('-')+1: len(date)-1])
            #controllo se devo aggiornare
            dtonline = dt.datetime(daa, dmm, dgg)
            if not rdate > dtonline:
                res = scraping(up_link)
                table_write(res, worksheet, row)
```

4.8 – Funzione *swap*

4.8.1 – Introduzione

La funzione *swap* serve modificare l'ordine dei record contenuti in *links.txt*. Essa sposta in fondo al file di testo l'elemento passato come argomento.

4.8.2 – Funzionamento

La funzione inizialmente recupera tutti gli elementi contenuti in *links.txt* e li mette in una variabile di tipo lista denominata *old_links*. Dopodiché inserisce in coda alla lista il link indicato da *sw_link* effettuando prima la sua rimozione e poi il suo reinserimento. Infine, sovrascrive *links.txt* con gli elementi di *old_links*.

4.8.3 – Codice

```
def swap(sw_link):
    """
    Funzione che sposta in fondo un link presente in links.txt.
```

```
    parametri:
        @type sw_link: str
        @param sw_link: il link da scambiare di posizione
    return:
        None
    """
    full_path = os.path.join(absolute_path, links_path)
    old_links = [r.rstrip('\n') for r in open(full_path, encoding='utf-8')]
    old_links.pop(old_links.index(sw_link))
    old_links.append(sw_link)
    #cancello il contenuto
    with open(full_path, 'r+', encoding='utf-8') as file:
        file.seek(0)
        file.truncate(0)
    #riscrivo
    for l in old_links:
        with open(full_path, 'a', encoding='utf-8') as file:
            file.write(l)
            file.write('\n')
```

4.9 – Funzione *logerror*

4.9.1 – Introduzione

La funzione *logerror* serve a salvare nel file *logerror.txt* gli errori che sono stati riscontrati durante il funzionamento del programma. Questi vengono trascritti come:
[giorno-mese-anno]: "oggetto_interessato_dall'errore" – Tipologia errore: "errore".

4.9.2 – Funzionamento

La funzione apre il file *logerror.txt,* salva la data corrente (attraverso la libreria *datetime*) e compone il messaggio d'errore (come descritto nell'introduzione) usando le due variabili passate per argomento. Queste due variabili, entrambe di tipo stringa, contengono reciprocamente l'oggetto dell'errore (*er_link*) e la tipologia d'errore (*exception*).

4.9.3 – Codice

```
def logerror(er_link, exception):
    """
    Funzione per il log degli errori avvenuti nello scraping. Salvataggio nel
file logerror.txt.

    parametri:
        @type er_link: str
```

```
        @param er_link: contiene il link della pagina interessata da errore
        @type exception: str
        @param exception: contiene l'eccezione lanciata
    return:
        None
    """
    absolute_path = os.path.dirname(__file__)
    full_path = os.path.join(absolute_path, logerror_path)
    date = dt.datetime.now()
    now = '[' + str(date.day) + '-' + str(date.month) + '-' + str(date.year)+']:
'
    with open(full_path, 'a', encoding='utf-8') as f1:
        f1.write(now + str(er_link) + ' - Tipologia errore: '+str(exception))
        f1.write('\n')
```

4.10 – Funzione *remove_link*

4.10.1 – Introduzione

La funzione *remove_link* serve per rimuovere un record contenuto in *links.txt*.

4.10.2 – Funzionamento

La funzione apre in lettura il file *links.txt* e ne salva il contenuto in una variabile denominata *filedata*. Poi sostituisce l'elemento contenuto nella variabile *rm_link* più il carattere a capo "\n" con una stringa vuota. Infine, sovrascrive il contenuto di *links.txt* con quello contenuto in *filedata*.

4.10.3 – Codice

```
def remove_link(rm_link):
    """
    Funzione che rimuove un link dal file links.txt.

    parametri:
        @type rm_link: str
        @param rm_link: contiene il link da rimuovere
    return:
        None
    """
    absolute_path = os.path.dirname(__file__)
    filedata = ''
    full_path = os.path.join(absolute_path, links_path)
    with open(full_path, 'r', encoding='utf-8') as file:
        filedata = file.read()
```

```
    filedata = filedata.replace(str(rm_link)+'\n', '')
    with open(full_path, 'r+', encoding='utf-8') as file:
        file.seek(0)
        file.truncate(0)
    with open(full_path, 'w', encoding='utf-8') as file:
        file.write(filedata)
```

4.11 - *BeautifulSoupException*

4.11.1 – Introduzione

BeautifulSoupException è una eccezione che è stata creata per poter essere invocata quando la risposta del server all'interrogazione da parte del programma non è "HTTP 200 OK".

4.11.2 - Codice

```
class BeautifulSoupException(Exception):
    """
    Eccezione che viene lanciata quando non la risposta del server non è
"HTTP 200 OK."
    """
    def __init__(self, link, code, message = 'Link non valido') -> None:
        self.link = link
        self.code = code
        self.message = message
        super().__init__(self.message)
```

Conclusioni

Il programma *WebScraping.py* offre concretamente la possibilità di usare i dati che estrae per implementare un database terminologico, o per usarli come una base per la sua realizzazione in quanto:

1. Estrae tutti i dati rilevanti.
2. Non sono presenti doppioni poiché tiene traccia dei vocaboli che sono già stati salvati.
3. I dati sono sempre consistenti con quelli online dal momento che controlla la presenza di eventuali modifiche negli originali.
4. Ogni record è identificabile in modo univoco perché l'URL della pagina è diverso per ogni vocabolo.

Tuttavia, sono da segnalare alcune possibili criticità che sono:

1. La lentezza di alcune operazioni come il salvataggio e l'aggiornamento dei vocaboli, le quali in base al numero di dati da processare possono richiedere anche diverse ore.
2. Il salvataggio dei paragrafi di definizione dei vocaboli in unica riga potrebbe non essere adeguato a un utilizzo in un database terminologico poiché potrebbe essere necessaria una sua ulteriore suddivisione per poterli classificare in maniera più consona.

In conclusione, il programma *WebScraping.py* offre una solida base per l'implementazione di un database terminologico a fronte di alcune leggere modifiche ai dati salvati.

Bibliografia e sitografia

1. Software Engineering (10th Edition) - Sommerville Ian - PEARSON EDUCACION 2015
2. https://www.um.es/lexico-comercio-medieval/index.php/p/v/inicio
3. https://www.crummy.com/software/BeautifulSoup/bs4/doc/
4. https://requests.readthedocs.io/en/latest/
5. https://requests.readthedocs.io/en/latest/_modules/requests/exceptions/
6. https://openpyxl.readthedocs.io/en/stable/
7. https://docs.python.org/3/library/
8. https://epydoc.sourceforge.net/epytext.html
9. https://www.w3schools.com/python/
10. https://realpython.com/python-web-scraping-practical-introduction/
11. https://realpython.com/python-requests/
12. https://www.geeksforgeeks.org/python-writing-excel-file-using-openpyxl-module/
13. https://www.freecodecamp.org/italian/news/i-file-in-python-aprire-leggere-scrivere-file-e-altre-funzioni-di-gestione-file/
14. https://www.march.es/es