CHAPTER 5

FoCUS: Learning to Crawl Web Forums

In this chapter, This chapter contains five (5) sections explaining the experiment: Section 5.1 FoCUS Framework, Section 5.2 FoCUS System Block Diagram, Section 5.3 FoCUS Data Flow Diagram, Section 5.4 Flowchart, Section 5.5 Execute Section, 5.6 Evaluation and Discussion and Section, 5.7 Highlight of Contribution and Conclusion.

5.1 FoCUS Framework

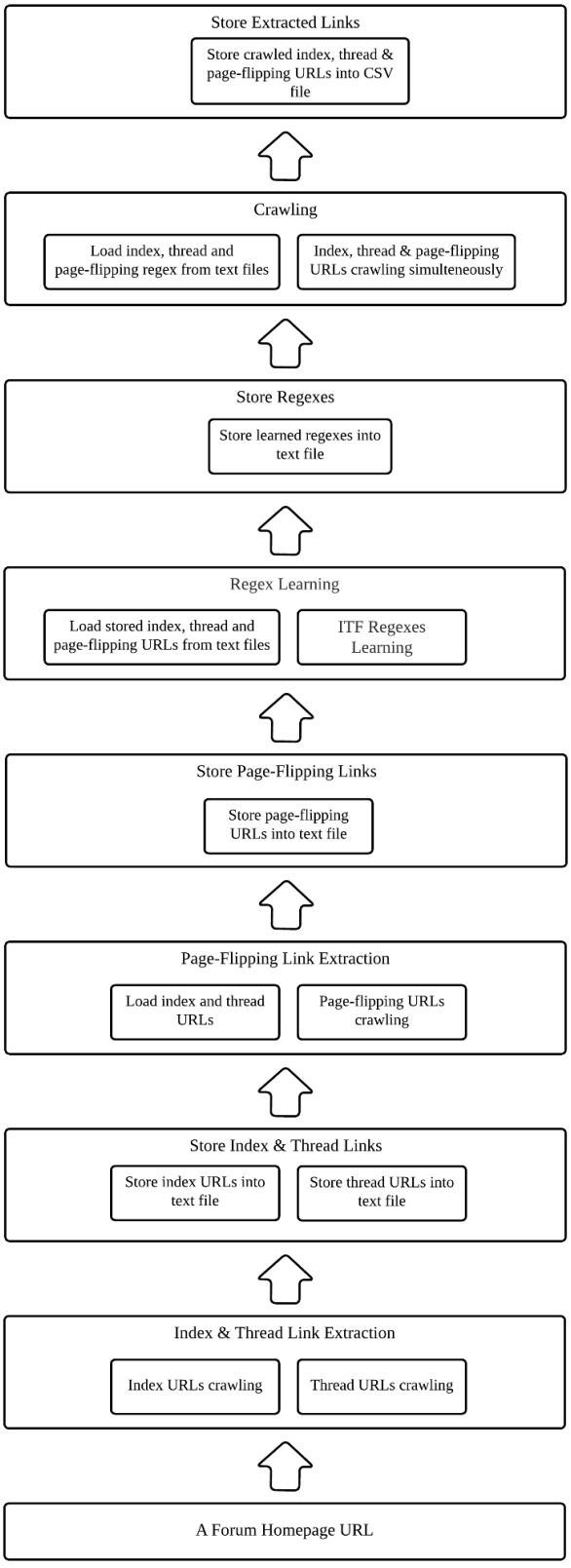


Figure 5.1: FoCUS Layered Framework

Figure 5.1 presents the design of the FoCUS layered framework that contains 9 layers. The 9 layers include A Forum Homepage URL, Link Extraction, Store Extracted Links, Regex Learning, Store Regexes, Crawling and Store Extracted Links. Each layer of this framework will handle different tasks that ensure the final results are achieved.

5.1.1 A Forum Homepage URL

First of all, the design of the flow of framework begins by accepting a web forum homepage URL. The web forum homepage URL is chosen because the homepage is the root page of all subsequent forum pages which means homepage is the entry page that is the lowest common ancestor of all index pages and thread pages in a web forum.

5.1.2 Index & Thread Link Extraction

After we get the homepage URL of a web forum, the program will start crawling the index URLs and thread URLs that exist in the web forum.

5.1.3 Store Index & Thread Links

After crawling the index and thread URLs, all the crawled index URLs and thread URLs will be saved in 2 different text files respectively.

5.1.4 Page-Flipping Link Extraction

At this layer, the program will first retrieve all the saved index and thread URLs from the text files that we have created just now to continue to crawl all the page-flipping URLs that exist within these index and thread URLs.

5.1.5 Store Page-Flipping Links

After crawling the page-flipping URLs, all the crawled page-flipping URLs will be saved in a text file as well.

5.1.6 Regex Learning

At this layer, the program will first retrieve all the saved index, thread and page-flipping URLs from the text files and then learn the link patterns of an index URL, thread URL and page-flipping URL. The program will then create the regex pattern for index URL, thread URL as well as page-flipping URL

5.1.7 Store Regexes

After creating the regex pattern for index URL, thread URL and page-flipping URL, all these regex patterns will be saved in a text file.

5.1.8 Crawling

At this layer, there will be another crawler (can be known as FoCUS crawler) we have to use to crawl all the index, thread and page-flipping URLs, which we need to retrieve all the regex patterns that we have just saved in the text file and feed it to this crawler. Then, we need to pass a homepage URL of a forum again to this crawler, it will then automatically crawl all index, thread and page-flipping URLs of the web forum that we have passed in.

5.1.9 Store Extracted Links

After the crawler is done crawling, all the crawled index, thread and page-flipping URLs will be saved in a CSV file.

5.2 FoCUS System Block Diagram

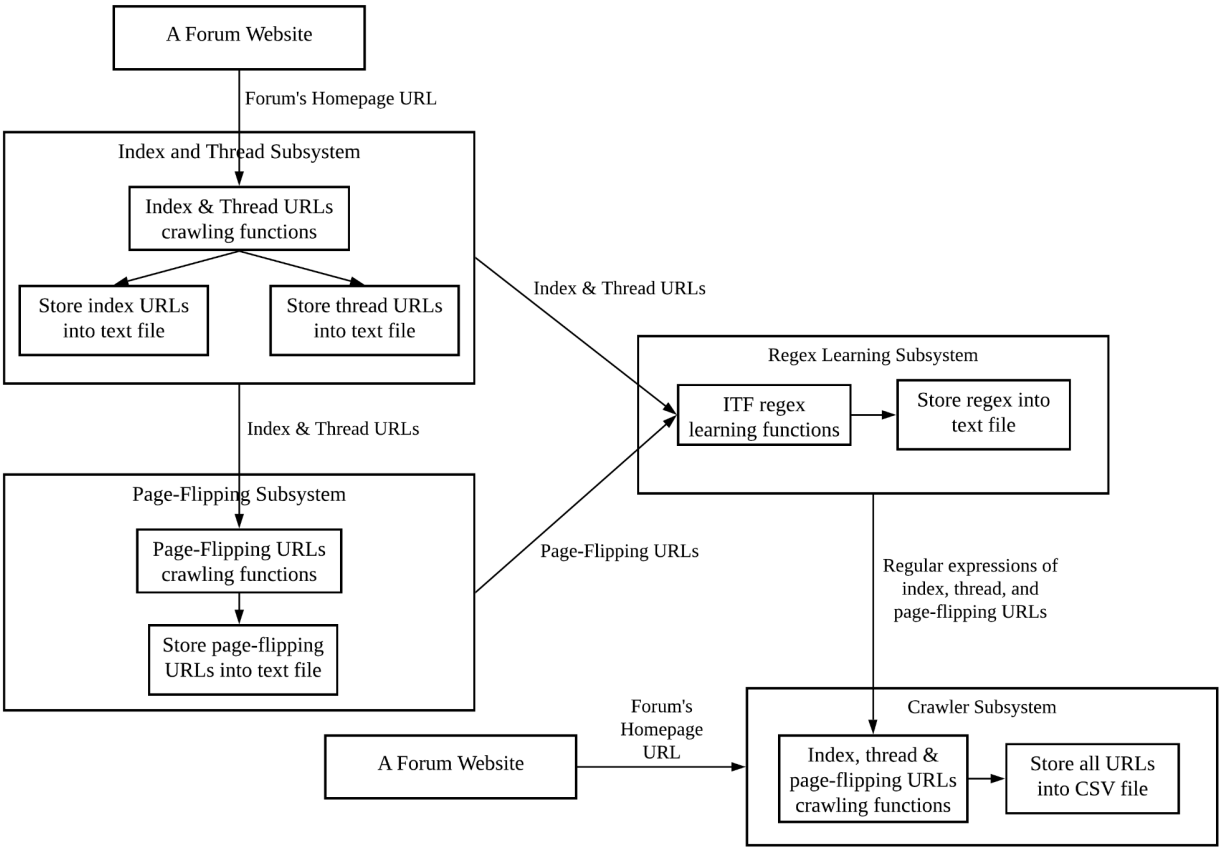


Figure 5.2: System Block Diagram

Figure 5.2 presents the system block diagram for the FoCUS framework. From the diagram above, we can see that there are a total of 4 subsystems to form this FoCUS framework. The subsystems are *Index and Thread Subsystem*, *Page-Flipping Subsystem*, *Regex Learning Subsystem* and *Crawler Subsystem*.

5.2.1 Index and Thread Subsystem

First of all, this subsystem will accept a homepage url from a forum website. Therefore, we have used the homepage of SixCrazyMinutes forum website (http://www.sixcrazyminutes.com/forums) to act as the entry page and pass it into the system. After the homepage url of a forum website is passed in, this subsystem will start crawling index and thread URLs that present in this forum. After crawling, all the index URLs and thread URLs will be stored in a different text file respectively, namely *index_url.txt* and *thread_url.txt*.

5.2.2 Page-Flipping Subsystem

After that, the page-flipping subsystem will retrieve all index URLs and thread URLs stored in index.txt and thread.txt respectively. After retrieving URLs from the text files, this subsystem will start to crawl all the page-flipping URLs that may exist in each index and thread URL just retrieved. When done crawling, all the page-flipping URLs will be stored in a text file as well, namely *page_flipping_url.txt*.

5.2.3 Regex Learning Subsystem

Later, the regex learning subsystem will retrieve all index URLs, thread URLs and page-flipping URLs that have been crawled and stored in different text files just now. After retrieving URLs from the text files, this subsystem will start learning the patterns of an index URL, thread URL and page-flipping URL. Then, it will find out what the regex pattern of an index URL link, thread URL link and page-flipping URL link look like and the regular expression patterns of these three different types of URL links will also be saved in a text file, namely *URL_Regex.txt*.

5.2.4 Crawler Subsystem

Finally, the forum crawler subsystem comes into the role. This subsystem is the crawler that is built using a free and open-source web-crawling framework called Scrapy. It first will retrieve all the regular expressions of index, thread and page-flipping URLs and utilize these regular expressions to crawl all the index links, thread links, and page-flipping links of a forum website. Before this, we have used the SixCrazyMinutes forum website to train all the necessary regular expressions, so now we can pass in the homepage url of the SixCrazyMinutes web forum again to this crawler and this crawler will automatically help to crawl all of the index links, thread links and page-flipping links that exist in this forum website as much as possible and saved the results into a CSV file called *FinalOutput.csv*. The good thing about this crawler is that it can crawl other web forums as well if other web forums' link structures are similar to the SixCrazyMinutes forum. For example, this crawler also successfully crawled https://www.namepros.com/, https://www.gardenstew.com/ and so on. The user just needs to enter the homepage URL of these

forums and the crawler will directly crawl all of the index, thread and page-flipping URLs based on the regular expressions learned.

5.3 FoCUS Data Flow Diagram (DFD)

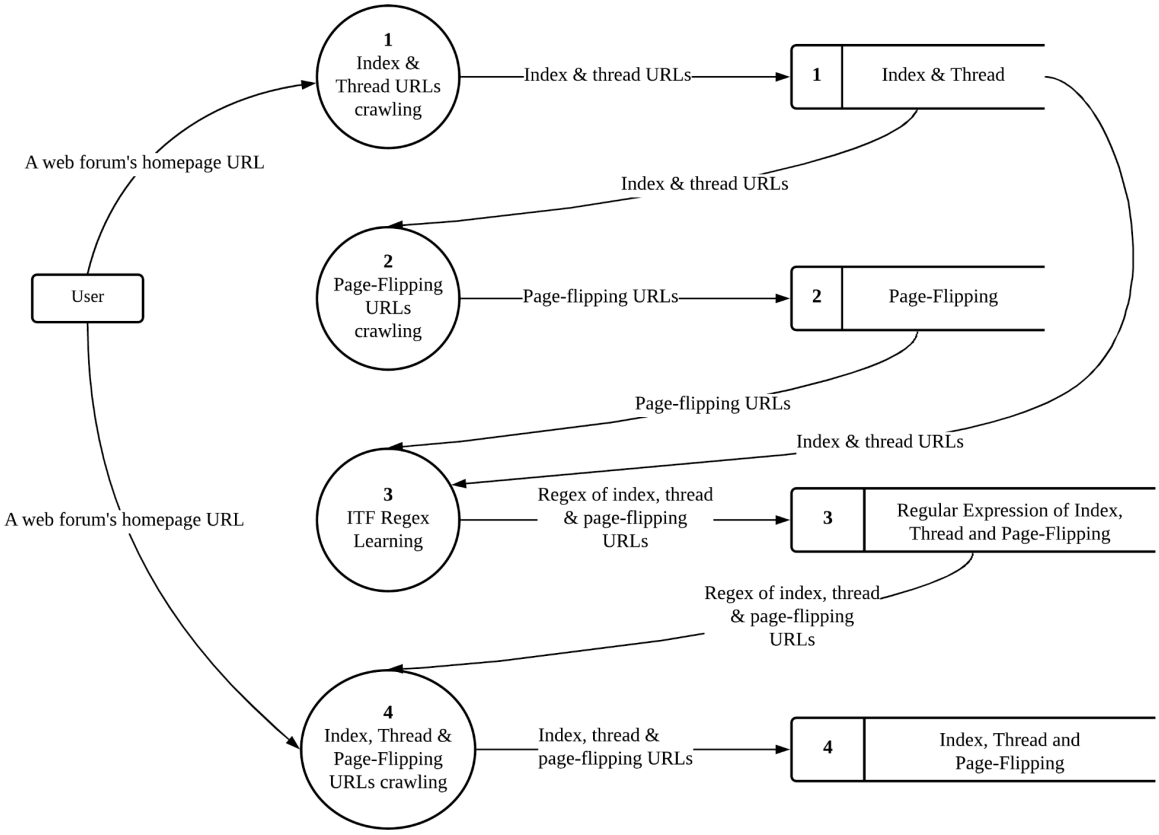


Figure 5.3: FoCUS Data Flow Diagram

Figure 5.3 presents the data flow diagram for the FoCUS web forum crawler. From the diagram above, we can see that there is one external entity, 4 processes and 4 data stores. Below is the diagram that describes the meaning of each shape in the data flow diagram.

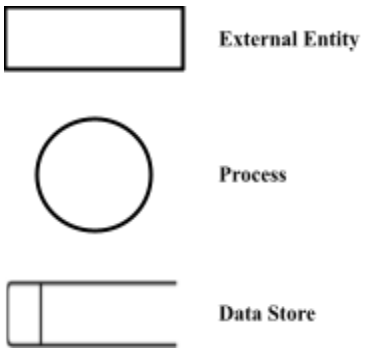


Figure 5.4: Shape meaning in Data Flow Diagram

5.3.1 External Entity: User

Moving forward, we can see that there is an **external entity** called **user** in the data flow diagram. User is the one who uses this program or system to crawl data from a web forum. First of all, the user is required to enter or provide the homepage URL of the web forum that he/she wants to crawl to the program, which the web forum's homepage URL will be passed to the **process 1**. For example, the homepage of SixCrazyMinutes forum website (http://www.sixcrazyminutes.com/forums)

5.3.2 Process 1: Index & Thread URLs crawling

In **Process 1**, the program will receive the web forum's homepage url provided by the user. It will then start crawling all the index and thread URLs that exist in the web forum. The program will crawl all the index URLs first and then crawl all the thread URLs that exist in each index URLs that have just crawled.

5.3.3 Data Store 1: Index & Thread

After the **process 1** is done, the **first data store** comes into play, where all index URLs and thread URLs will be stored in a different text file respectively, namely *index_url.txt* and *thread_url.txt*.

5.3.4 Process 2: Page-Flipping URLs crawling

In **Process 2**, the program will retrieve the crawled results of process 1 from the data store 1. After retrieving all the index and thread URLs, the program will start to crawl all the page-flipping URLs that may exist in these index and thread URLs.

5.3.5 Data Store 2: Page-Flipping

After the **process 2** is done, the **second data store** comes into play, where all the page-flipping URLs will be stored in a text file, namely *page_flipping_url.txt*.

5.3.6 Process 3: ITF Regex Learning

In **Process 3**, the program will retrieve all index URLs, thread URLs and page-flipping URLs from data store 1 and 2. Then, it will start learning the patterns of an index URL, thread URL and page-flipping URL, which it will find out what the regex pattern of an index URL link, thread URL link and page-flipping URL link look like and create regular expressions for each of these URLs.

5.3.7 Data Store 3: Regular Expression of Index, Thread and Page-Flipping

After the **process 3** is done, the **third data store** comes into play, where the regular expression patterns of index, thread and page-flipping URLs will be stored in a text file called *URL_Regex.txt*.

5.3.8 Process 4: Index, Thread & Page-Flipping URLs crawling

In **Process 4**, the program will retrieve all the regular expressions of index URLs, thread URLs and page-flipping URLs from data store 3 and utilize these regular expressions to crawl all the index links, thread links, and page-flipping links of a forum website. Before this, we have used the SixCrazyMinutes forum website to train all the necessary regular expressions, so now we can pass in the homepage url of the SixCrazyMinutes web forum again to this crawler and then this crawler will automatically help the user to crawl all of the index links, thread links and page-flipping links that exist in this forum website as much as possible. The user also can crawl other web forums as well if other chosen web forums' link structure are similar to the SixCrazyMinutes forum. For example, this crawler also successfully crawled the forum https://www.namepros.com/, https://www.gardenstew.com/ and so on. The user just needs to enter the homepage URL of these forums and the crawler will directly go to **process 4** to crawl all of the index, thread and page-flipping URLs based on the regular expressions learned.

5.3.9 Data Store 4: Index, Thread & Page-Flipping

After the **process 4** is done, the **forth data store** comes into play, where all of the index, thread and page-flipping URLs of a web forum will be saved in a CSV file called *FinalOutput.csv*.

5.4 Flowchart

In this project, the FoCUS crawler is made up of 5 python scripts. The 5 python scripts are:

1. **main.py** (python script that gather the main functions in other python scripts)
2. **Index_Thread.py** (python script that used to crawl all the index and thread links)
3. **Page_Flipping.py** (python script that used to crawl all the page-flipping links)
4. **ITF_Learning.py** (python script that used to learn the link structure of index, thread and page-flipping and create regular expression)
5. **ForumCrawler.py** (python script that used to crawl all of the index, thread and page-flipping links using the regex learned, known as FoCUS Crawler).

5.4.1 main.py

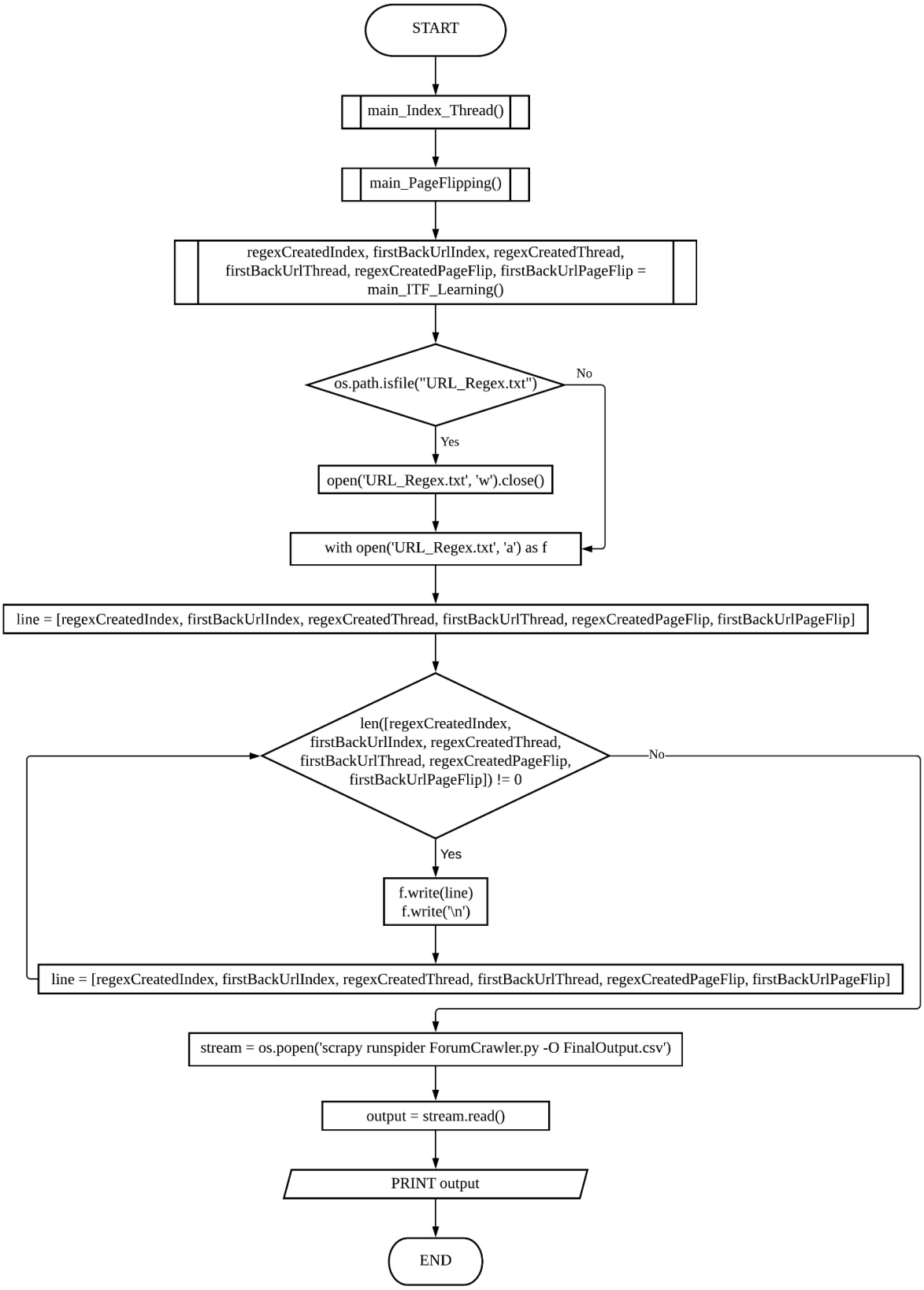


Figure 5.5: Flowchart of main.py

The above figure shows the program flow of the *main.py* script. We can see that the *main.py* calls the other main functions that exist in other python scripts. For example, the *main.py* script calls the `main_Index_Thread()`, `main_PageFlipping()` and `main_ITF_Learning()` functions one after another in the beginning. After crawling and learning, *the main.py* script will store the regular expressions returned by `main_ITF_Learning()` into a text file called *URL_Regex.txt*. After that, it will open up the terminal automatically and use the command assigned to it to run *ForumCrawler.py* for crawling.

`main_Index_Thread()` is the main function of the *Index_Thread.py* script while `main_PageFlipping()` and `main_ITF_Learning()` are the main functions of the *Page_Flipping.py* and *ITF_Learning.py* scripts respectively. Please refer to **section 5.3.2** for **`main_Index_Thread()`** function, refer to **section 5.3.3** for **`main_PageFlipping()`** function, and **section 5.3.4 for `main_ITF_Learning()`** function.

1. main_Index_Thread()

Description	Handle the index & thread URLs crawling as well as save the crawled URLs into text files.
Method Type	static
Argument	Do not require argument
Return	Do not return anything
Example	`main_Index_Thread()`
Example of return value	Do not return anything

2. main_PageFlipping()

Description	Handle the page-flipping URLs crawling and save the crawled URLs into text files.
Method Type	static
Argument	Do not require argument
Return	Do not return anything
Example	`main_PageFlipping()`

Example of return value	Do not return anything

3. main_ITF_Learning()

Description	Handle the index, thread and page-flipping URLs regular expression learning and.
Method Type	static
Argument	Do not require argument
Return (`str`)	Return 6 different regular expressions in string type.
Example	`regexCreatedIndex, firstBackUrlIndex, regexCreatedThread, firstBackUrlThread, regexCreatedPageFlip, firstBackUrlPageFlip = main_ITF_Learning()`
Example of return value	`regexCreatedIndex:` ^([a-zA-Z0-9]+-[a-zA-Z0-9]+)+\.\d+$ `firstBackUrlIndex:` forums `regexCreatedThread:` ^([a-zA-Z0-9]+-[a-zA-Z0-9]+)+\.\d+$ `firstBackUrlThread:` threads `regexCreatedThread:` ^page-\d+$ `firstBackUrlThread:` (forums|threads)

5.4.2 Index_Thread.py

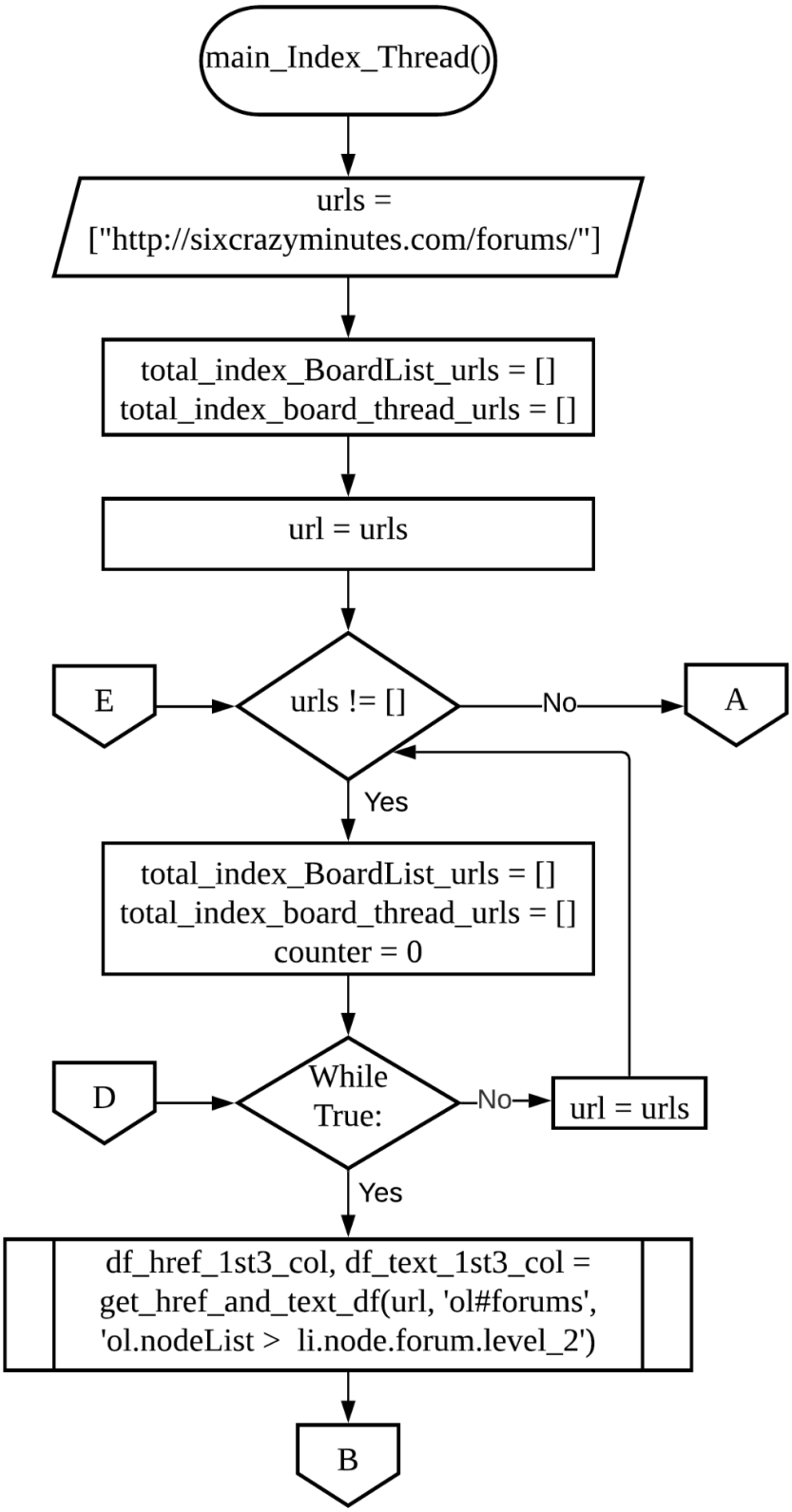


Figure 5.6: Flowchart of Index_Thread.py

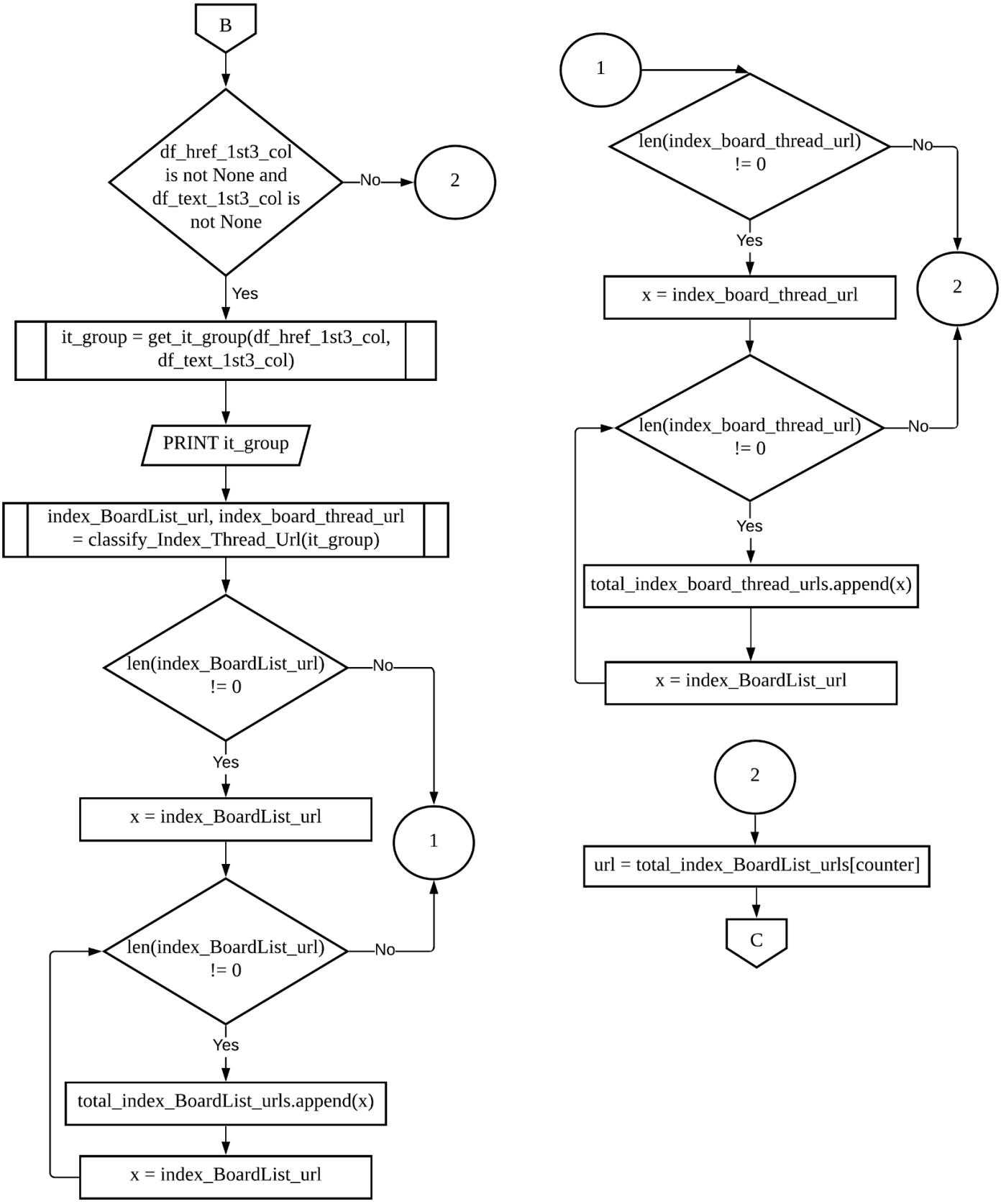


Figure 5.7: Flowchart of Index_Thread.py (continue)

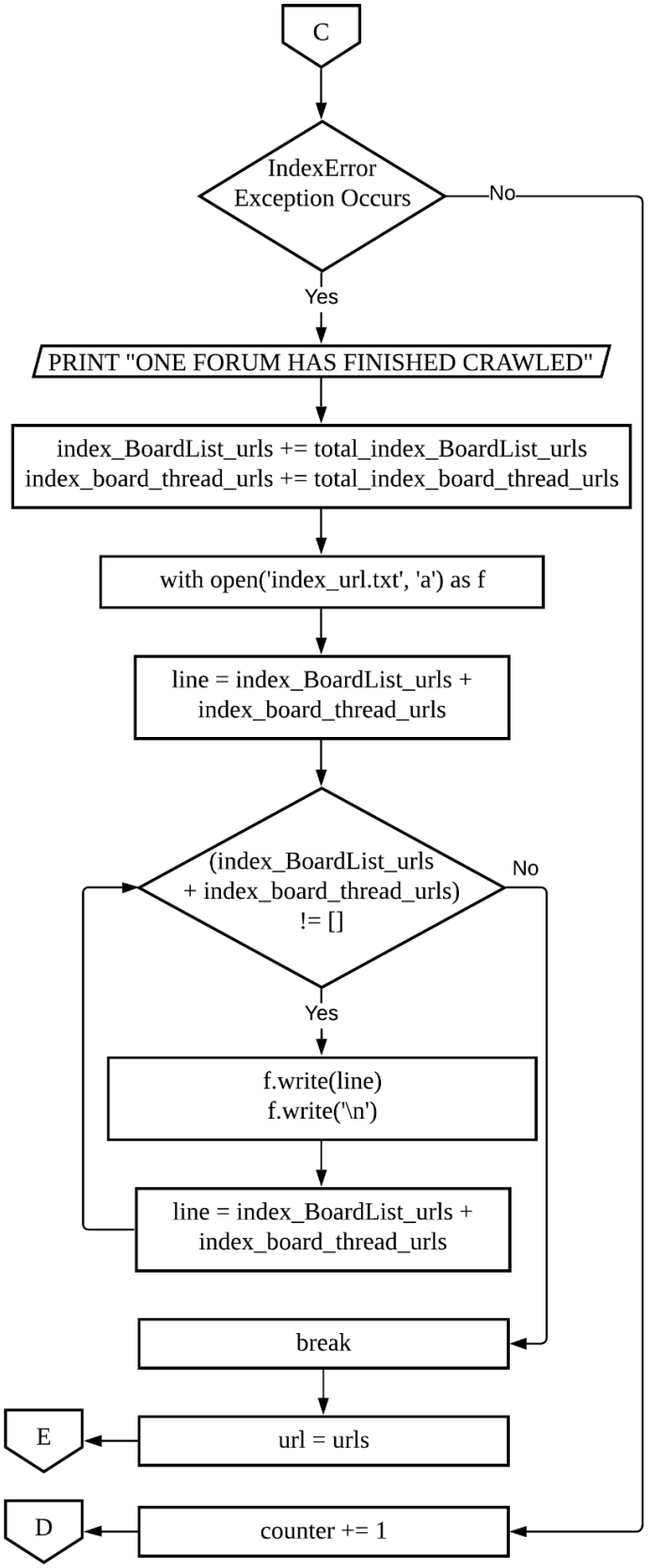


Figure 5.8: Flowchart of Index_Thread.py (continue)

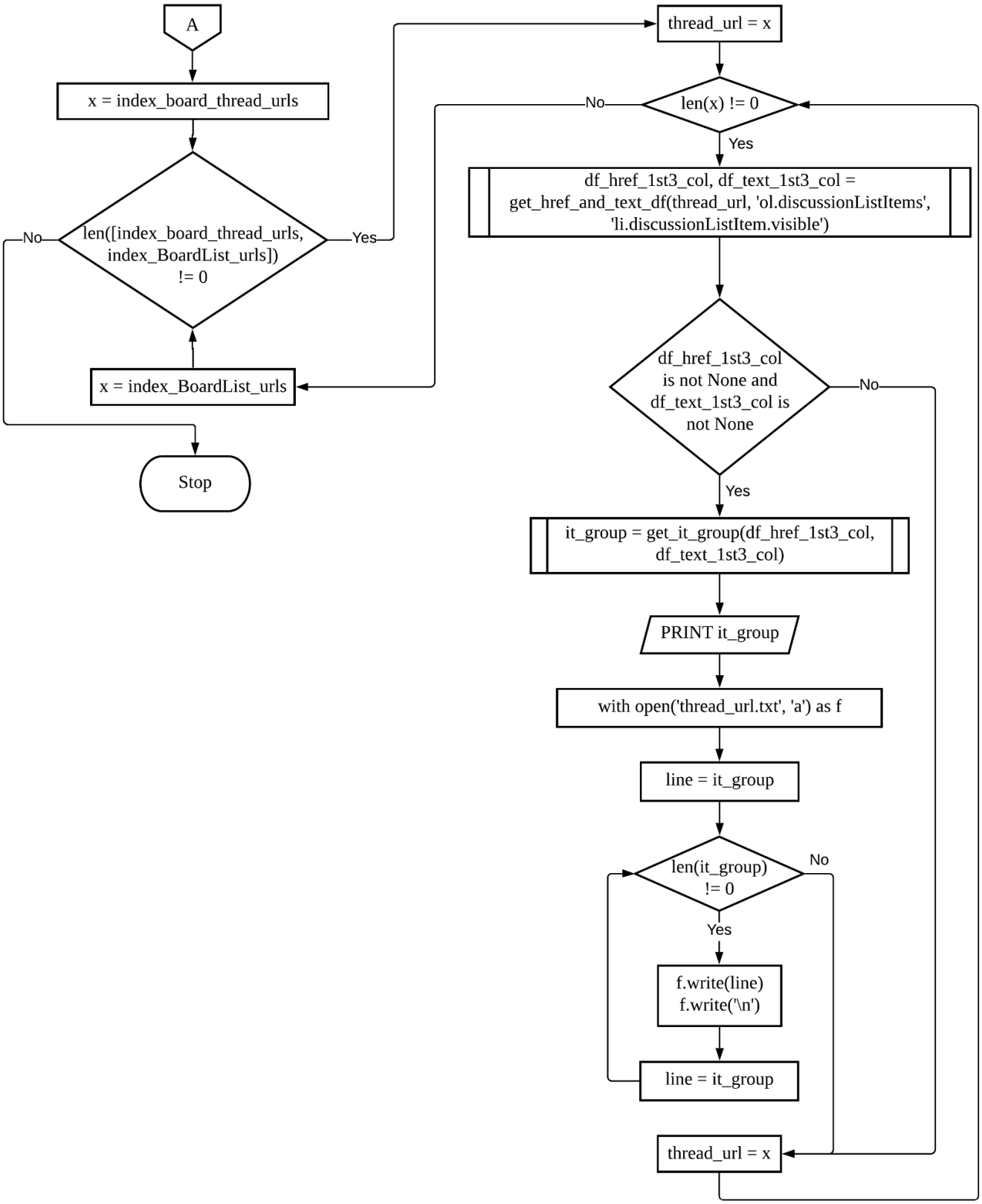


Figure 5.9: Flowchart of Index_Thread.py (continue)

5.4.2.1 Functions used in Index_Thread.py

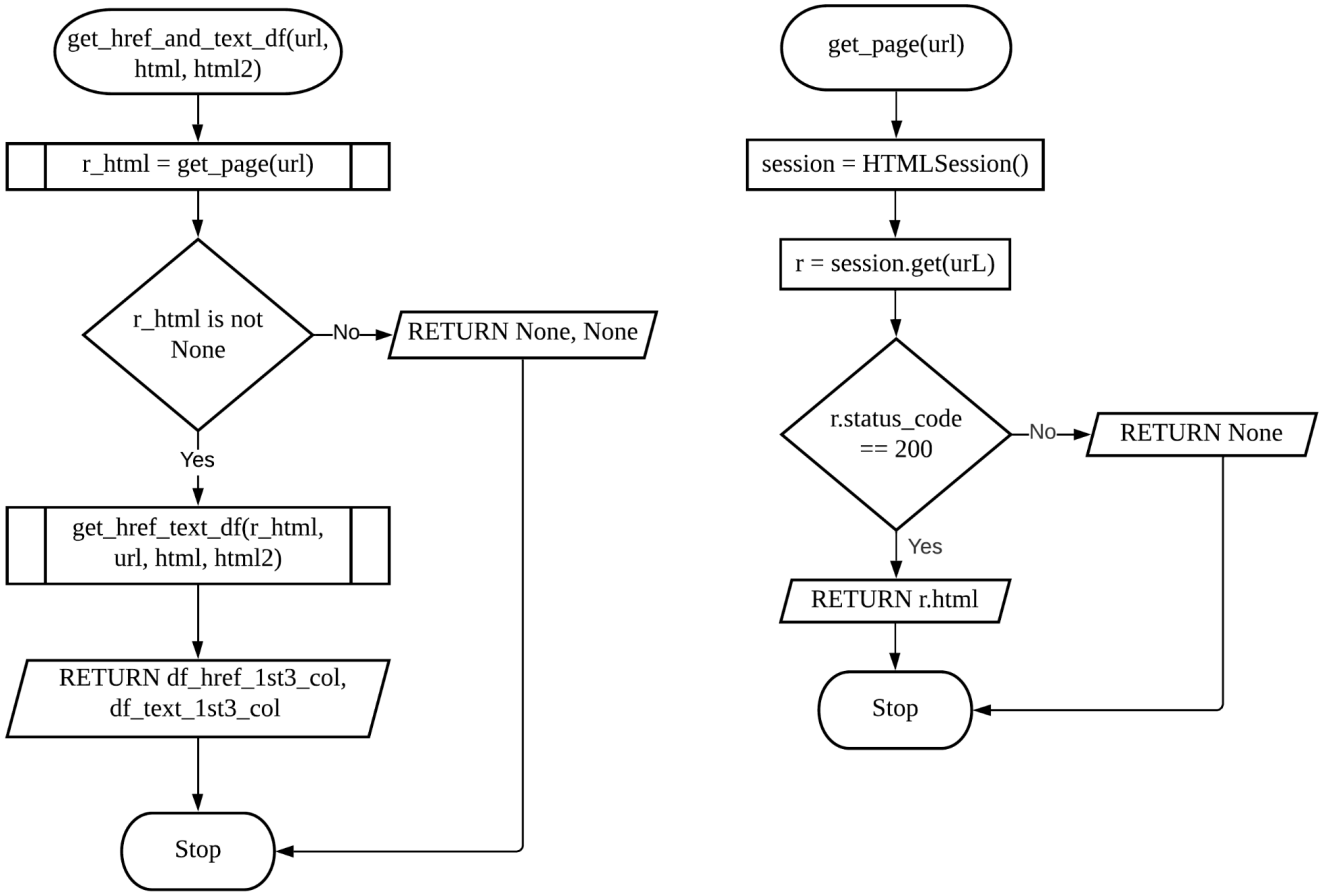
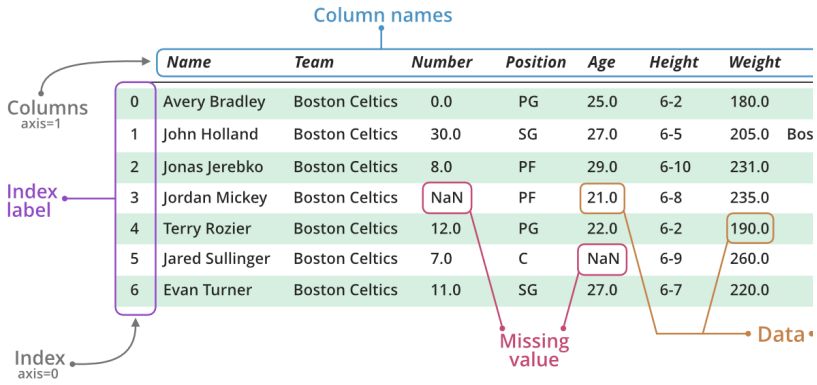


Figure 5.10: get_href_and_text_df() & get_page() functions

1. get_href_and_text_df(url, html, html2)

Description	Return 2 dataframes that contains 2 columns and having the same number of rows. `df_href_1st3_col` contains only links. `df_text_1st3_col` contains only anchor text. Each anchor text corresponds to the link in `df_href_1st3_col`.
Method Type	static
Argument	`url` = A web forum homepage's or entry page's URL `html` = html that wraps all different topics `html2` = html that represent each topic
Return (`pandas.DataFrame`)	Return 2 dataframes

Example	`df_href_1st3_col, df_text_1st3_col = get_href_and_text_df(url, 'ol#forums', 'ol.nodeList > li.node.forum.level_2')`
Example of return value	 **Reference only*

2. get_page(url)

Description	Use the provided `url` to load the website and then return the request-html object of the website.
Method Type	static
Argument	`url` = A web forum homepage's or entry page's URL
Return (`request-html`)	Return a request-html object that can be utilzed in scraping data on HTML document.
Example	`r_html = get_page(url)`
Example of return value	`r_html: <HTML url='http://www.sixcrazyminutes.com/forums/'>`

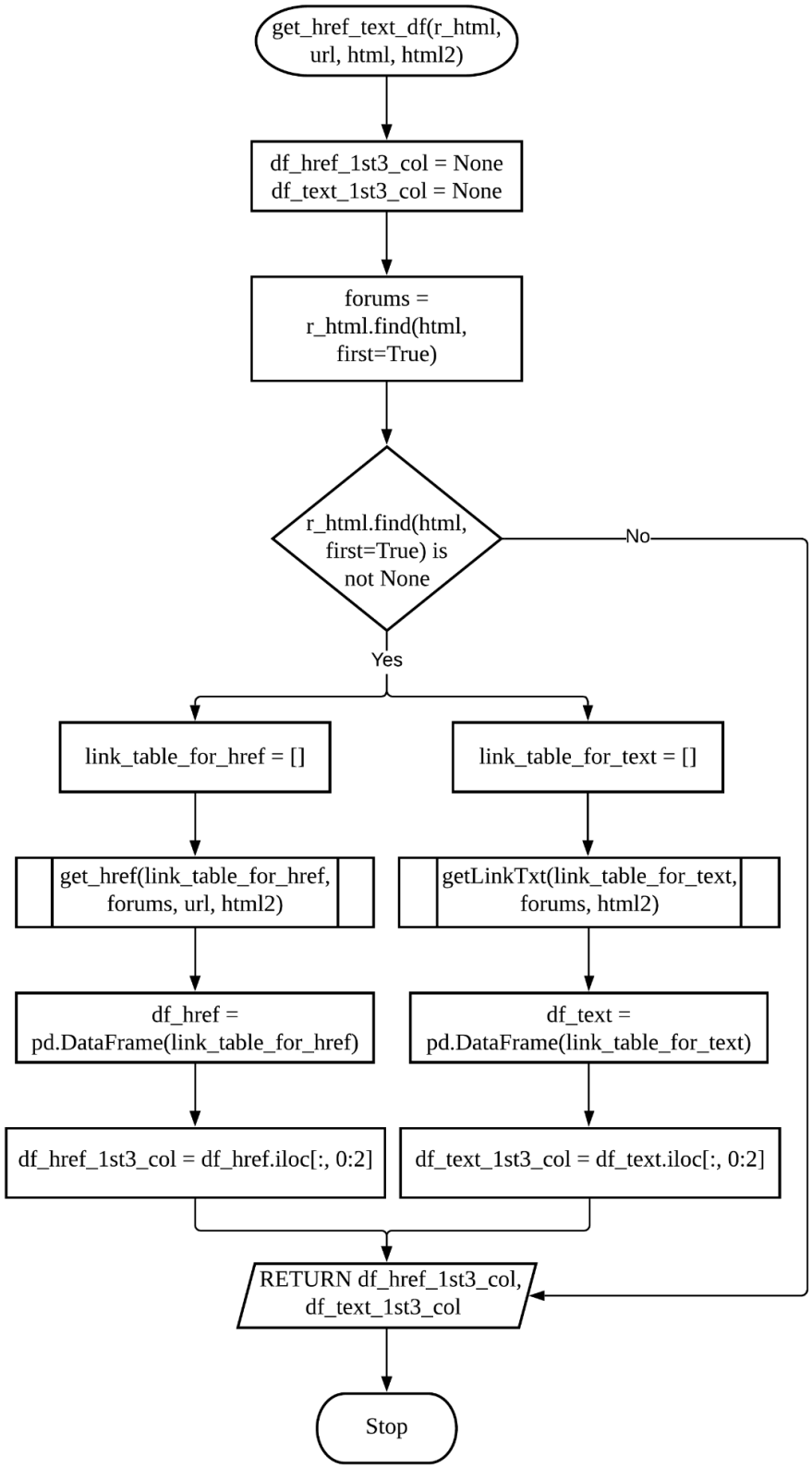
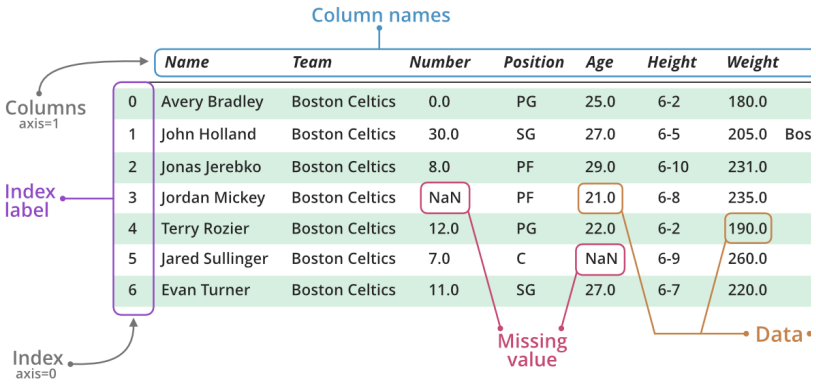


Figure 5.11: get_href_text_df() function

3. get_href_text_df(r_html, url, html, html2)

Description	Return 2 dataframes that contains 2 columns and having the same number of rows. `df_href_1st3_col` contains only links. `df_text_1st3_col` contains only anchor text. Each anchor text corresponds to the link in `df_href_1st3_col`.
Method Type	static
Argument	`r_html` = request-html object of a website `url` = A web forum homepage's or entry page's URL `html` = html that wraps all different topics on homepage `html2` = html that represent each topic on homepage
Return **(`pandas.DataFrame`)**	Return 2 dataframes
Example	`df_href_1st3_col, df_text_1st3_col = get_href_text_df(r_html, url, html, html2)`
Example of return value	 **Reference only*

	Name	Team	Number	Position	Age	Height	Weight	
0	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	
1	John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Bos
2	Jonas Jerebko	Boston Celtics	8.0	PF	29.0	6-10	231.0	
3	Jordan Mickey	Boston Celtics	NaN	PF	21.0	6-8	235.0	
4	Terry Rozier	Boston Celtics	12.0	PG	22.0	6-2	190.0	
5	Jared Sullinger	Boston Celtics	7.0	C	NaN	6-9	260.0	
6	Evan Turner	Boston Celtics	11.0	SG	27.0	6-7	220.0	

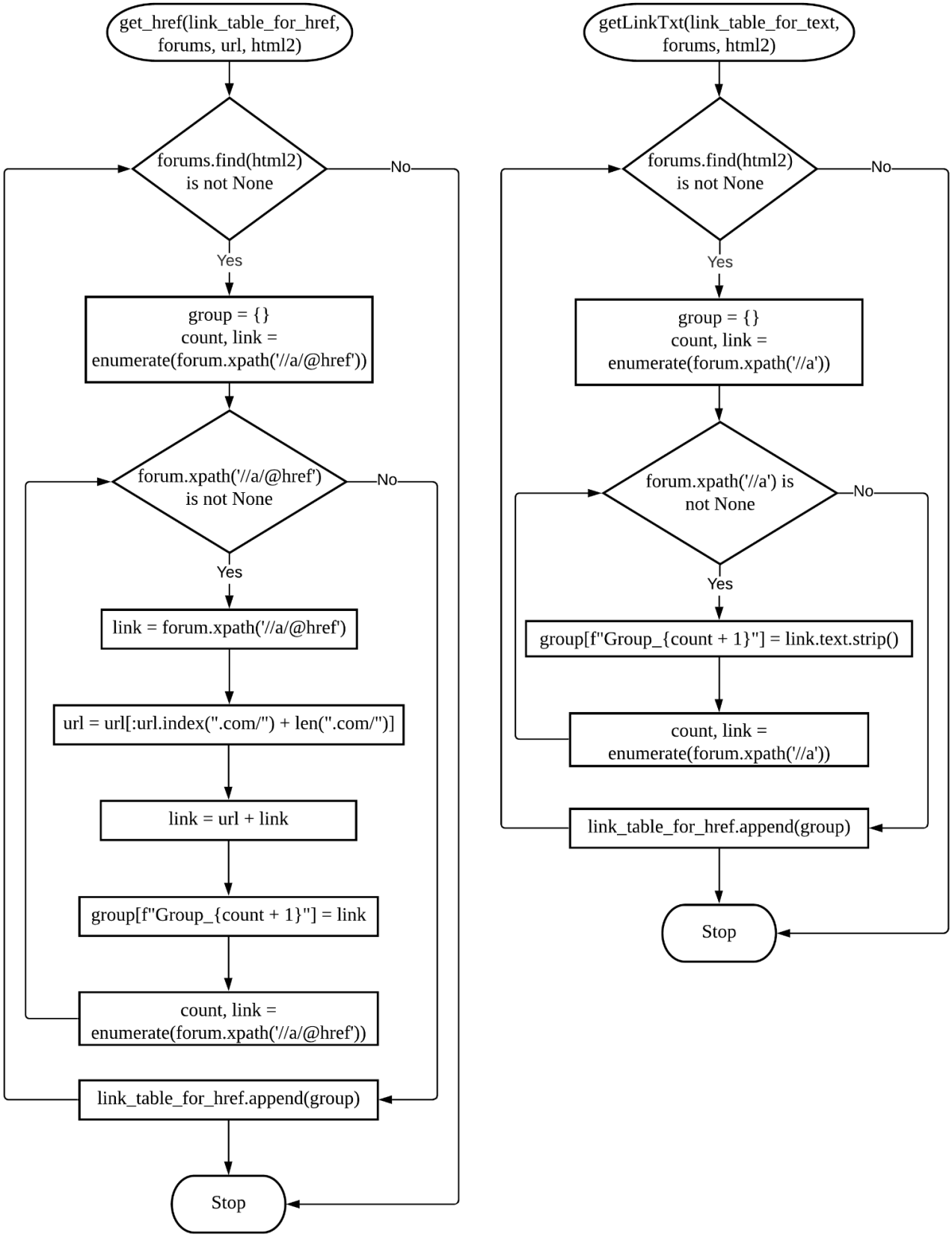


Figure 5.12: get_href() & getLinkTxt() functions

4. get_href(link_table_for_href, forums, url, html2)

Description	Append every crawled link into the `link_table_for_href` list.
Method Type	static
Argument	`link_table_for_href` = An empty list `forums` = A web element `url` = A web forum homepage's or entry page's URL `html2` = html that represent each topic on homepage
Return	Do not return anything
Example	`get_href(link_table_for_href, forums, url, html2)`
Example of return value	Do not return anything

5. getLinkTxt(link_table_for_text, forums, html2)

Description	Append every crawled anchor text into the `link_table_for_text` list.
Method Type	static
Argument	`link_table_for_text` = An empty list `forums` = A web element `html2` = html that represent each topic on homepage
Return	Do not return anything
Example	`getLinkTxt(link_table_for_text, forums, html2)`
Example of return value	Do not return anything

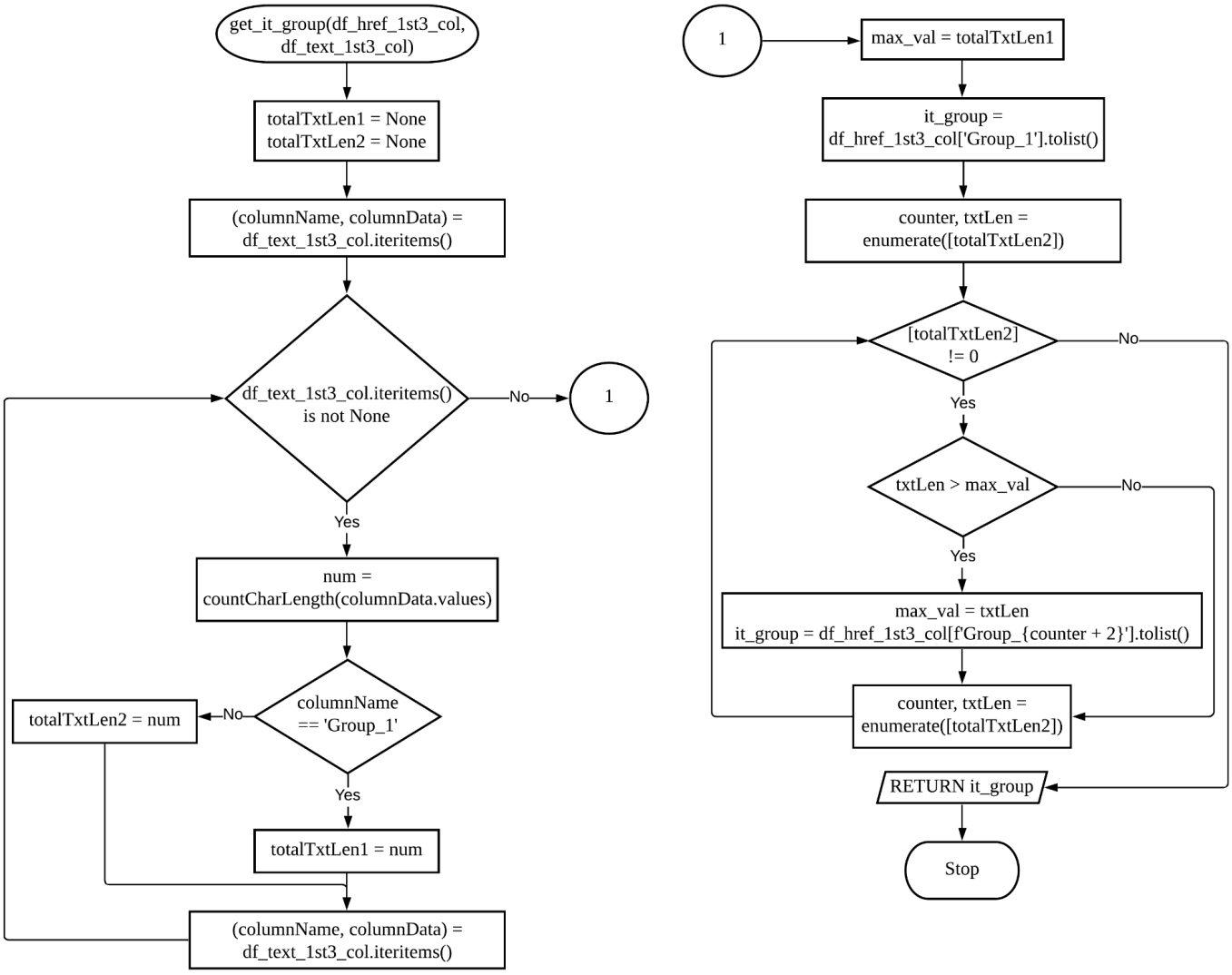


Figure 5.13: get_it_group() function

6. get_it_group(df_href_1st3_col, df_text_1st3_col)

Description	Find which column in the `df_text_1st3_col` dataframe has the highest number of words and then return that column in list type.
Method Type	static
Argument	`df_href_1st3_col` = Dataframes that contains 2 columns. The first column contains names like *Group_1, Group_2, Group_3, ...* and the second column contains index or thread links.

	`df_text_1st3_col` = Dataframes that contains 2 columns. The first column contains names like *Group_1, Group_2, Group_3, ...* and the second column contains anchor texts.
Return (`list`)	Return the column in `df_text_1st3_col` dataframe that has the highest number of words in list type.
Example	`it_group = get_it_group(df_href_1st3_col, df_text_1st3_col)`
Example of return value	`['http://sixcrazyminutes.com/forums/the-football-forum.4/', 'http://sixcrazyminutes.com/forums/liverpool-fc-news.32/', ...]`

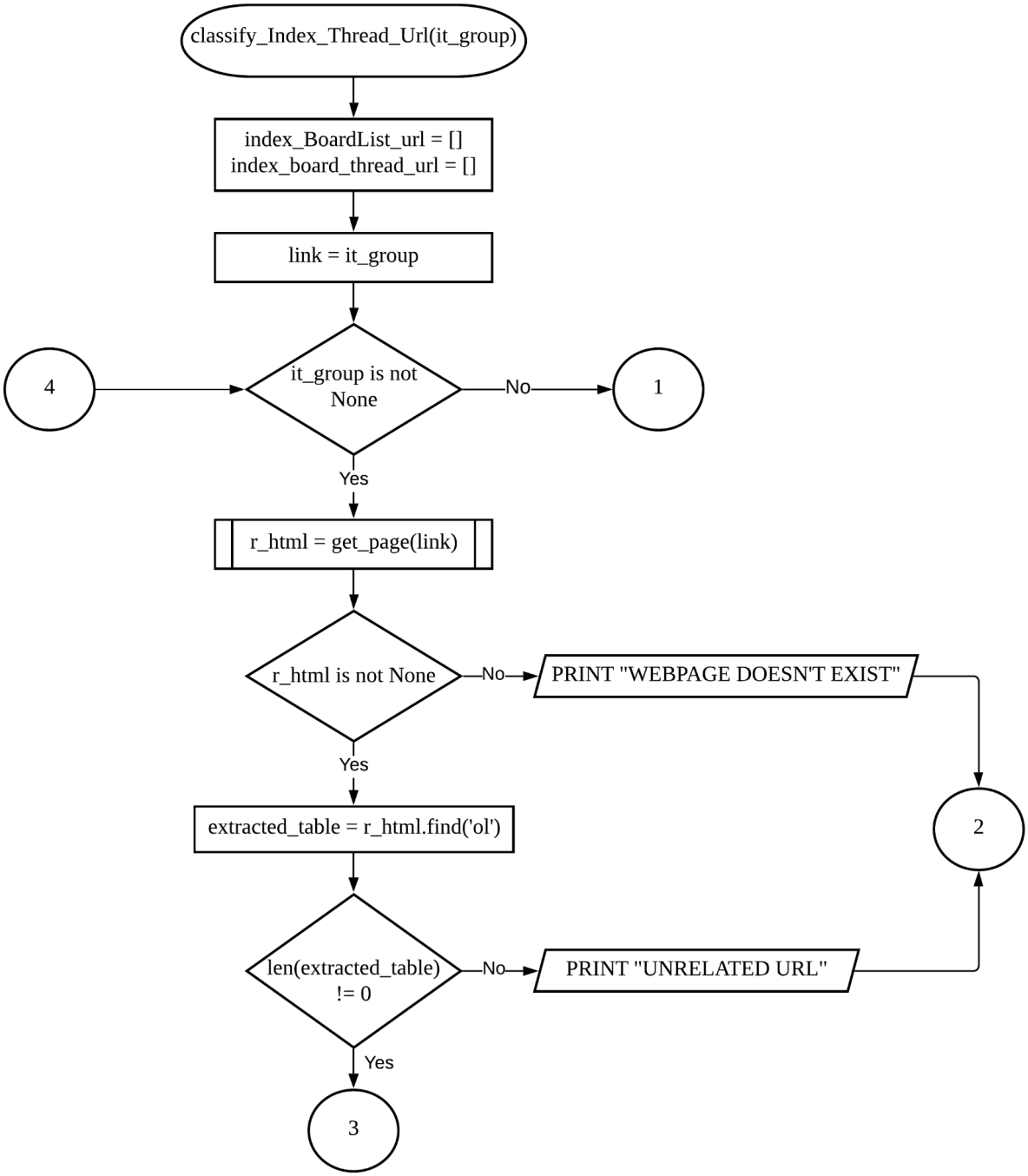


Figure 5.14: classify_Index_Thread_Url() function

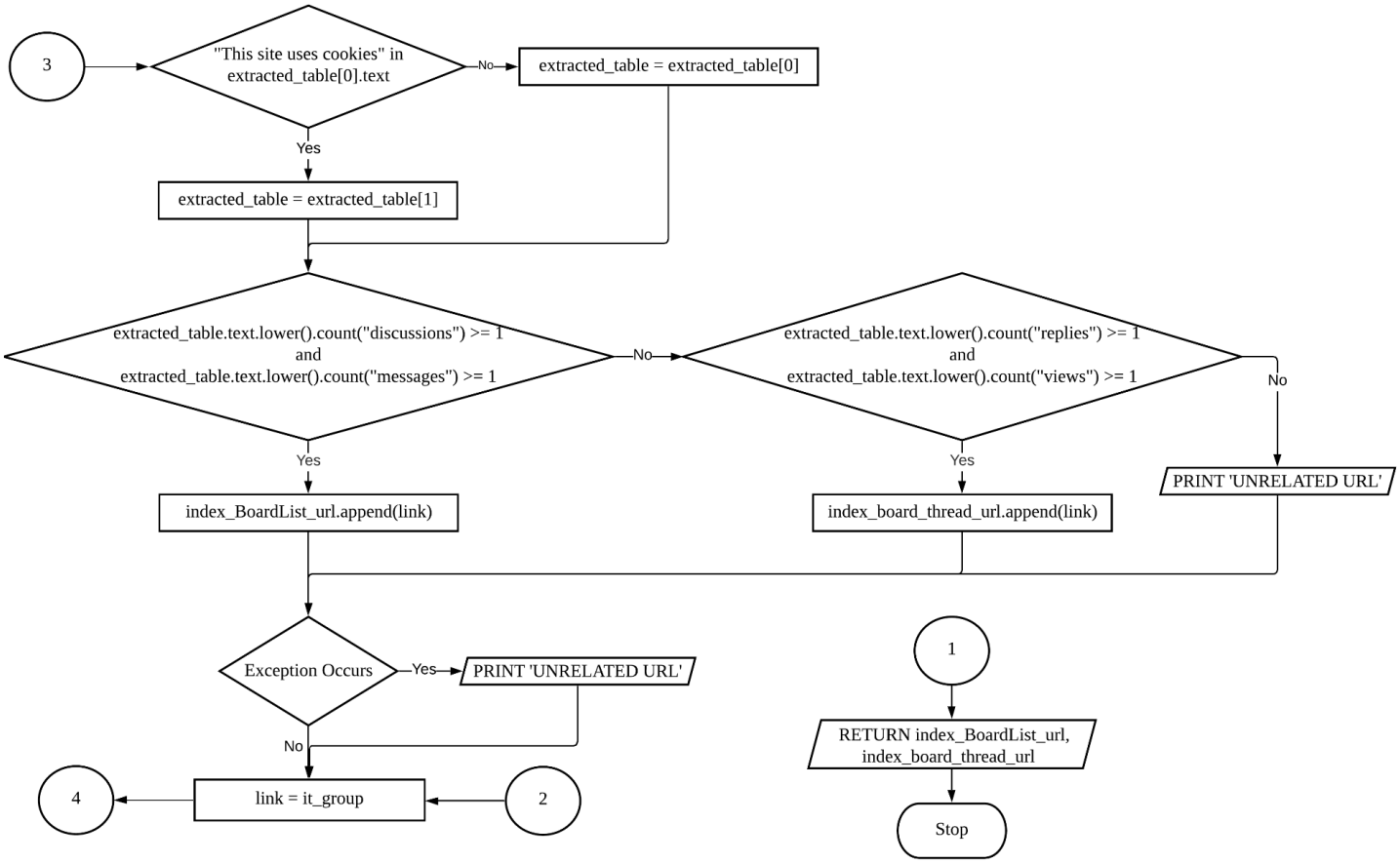


Figure 5.15: classify_Index_Thread_Url() function (continue)

7. classify_Index_Thread_Url(it_group)

Description	Classify the links in the `it_group`. If the link's destination page is a thread page, the link will be appended to the `index_board_thread_url` list. If the link's destination page is still an index page, the link will be appended to the `index_BoardList_url` list.
Method Type	static
Argument	`it_group` = A list that contains multiple index URLs or thread URLs
Return (`list`)	Return 2 lists variables, which are `index_BoardList_url` and `index_board_thread_url`. The first list will contain the links that their destination page is still an index page while

	the second list will contain the links that their destination page is a thread page.
Example	`index_BoardList_url, index_board_thread_url = classify_Index_Thread_Url(it_group)`
Example of return value	`['http://sixcrazyminutes.com/forums/the-football-forum.4/', 'http://sixcrazyminutes.com/forums/liverpool-fc-news.32/', ...]`

5.4.3 Page_Flipping.py

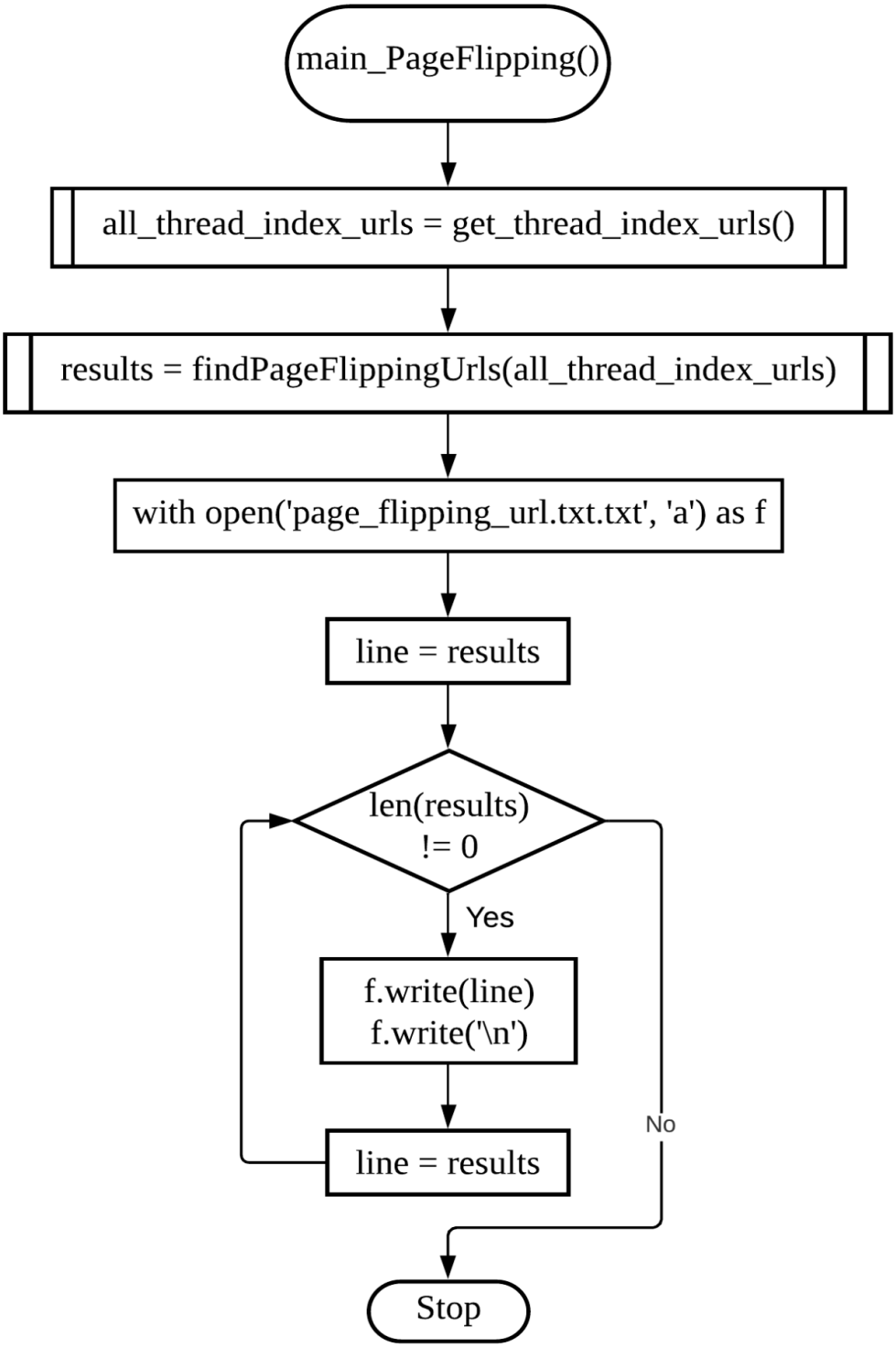


Figure 5.16: Flowchart of Page_Flipping.py

5.4.3.1 Functions used in Page_Flipping.py

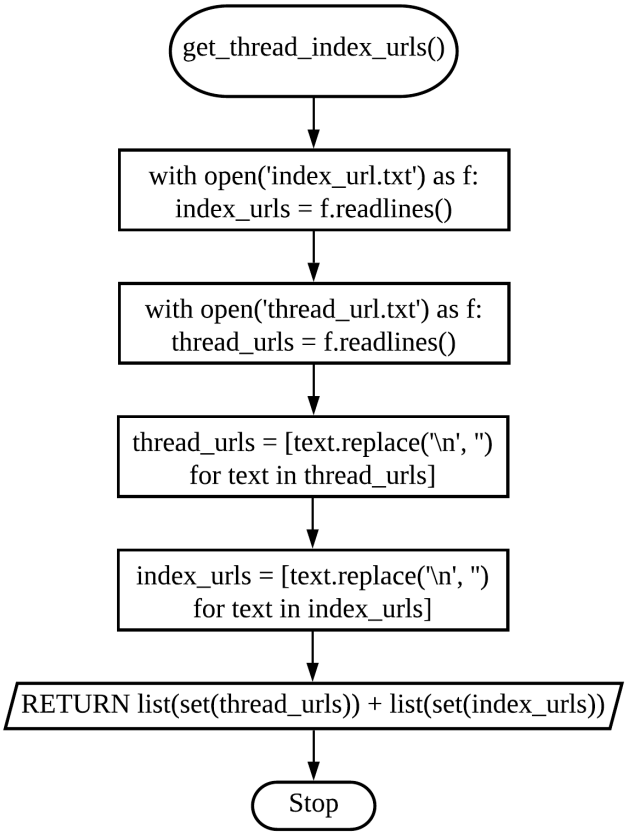


Figure 5.17: get_thread_index_urls() function

1. get_thread_index_urls()

Description	Retrieve all the links stored in *index_url.txt* and *thread_url.txt* text files by returning them in a list.
Method Type	static
Argument	Do not require argument
Return (`list`)	Return a list that contains index URLs and thread URLs
Example	`all_thread_index_urls = get_thread_index_urls()`
Example of return value	`['http://sixcrazyminutes.com/threads/username-missing-or-changes-needed.16626/', 'http://sixcrazyminutes.com/threads/new-feature-nsfw-options.29244/', ...]`

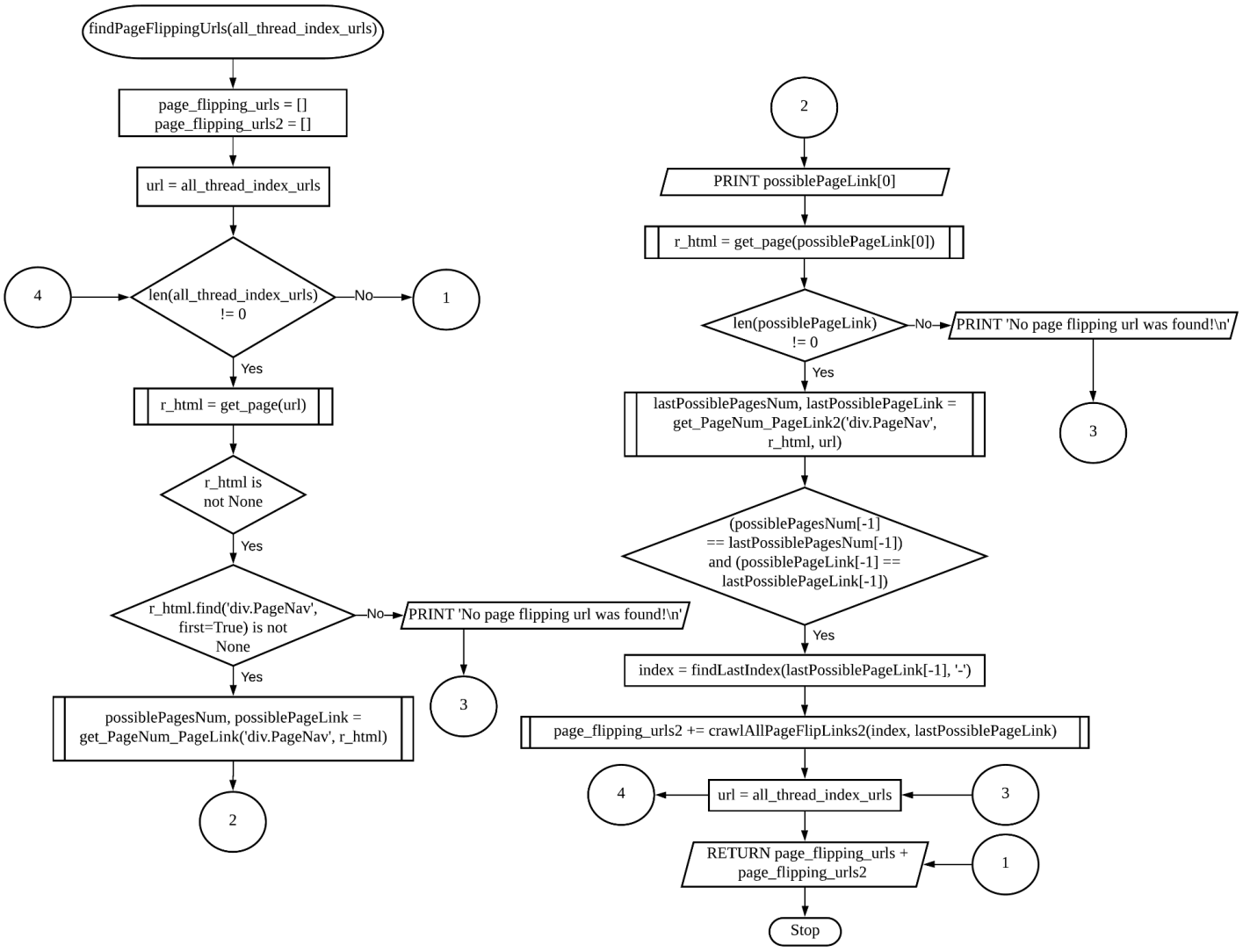


Figure 5.18: findPageFlippingUrls() function

2. findPageFlippingUrls(all_thread_index_urls)

Description	Crawl all the page-flipping URLs from the index and thread URLs received and then return the results in a list.
Method Type	static
Argument	`all_thread_index_urls` = A list that contains index URLs or thread URLs
Return (`list`)	Return a list that contains page-flipping URLs
Example	`results = findPageFlippingUrls(all_thread_index_urls)`

Example of return value	['http://sixcrazyminutes.com/threads/mods-adding-to-abuse-rather-than-stopping-it.113736/page-4', 'http://sixcrazyminutes.com/threads/mods-adding-to-abuse-rather-than-stopping-it.113736/page-3', ...]

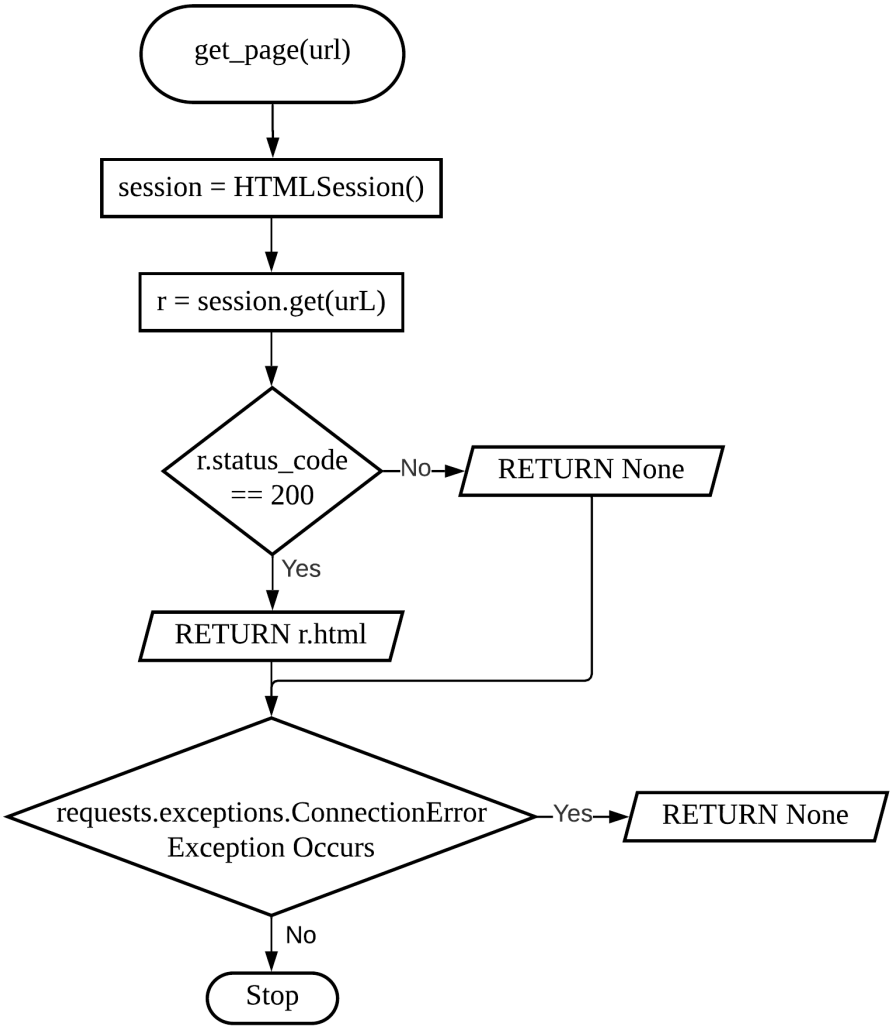


Figure 5.19: get_page() function

3. get_page(url)

Description	Use the provided `url` to load the website and then return the request-html object of the website.

Method Type	static
Argument	`url`= An index or a thread URL
Return (`request-html`)	Return a request-html object that can be utilzed in scraping data on HTML document.
Example	`r_html = get_page(url)`
Example of return value	`r_html: <HTML url='http://sixcrazyminutes.com/threads/mods-adding-to-abuse-rather-than-stopping-it.113736/page-4'>`

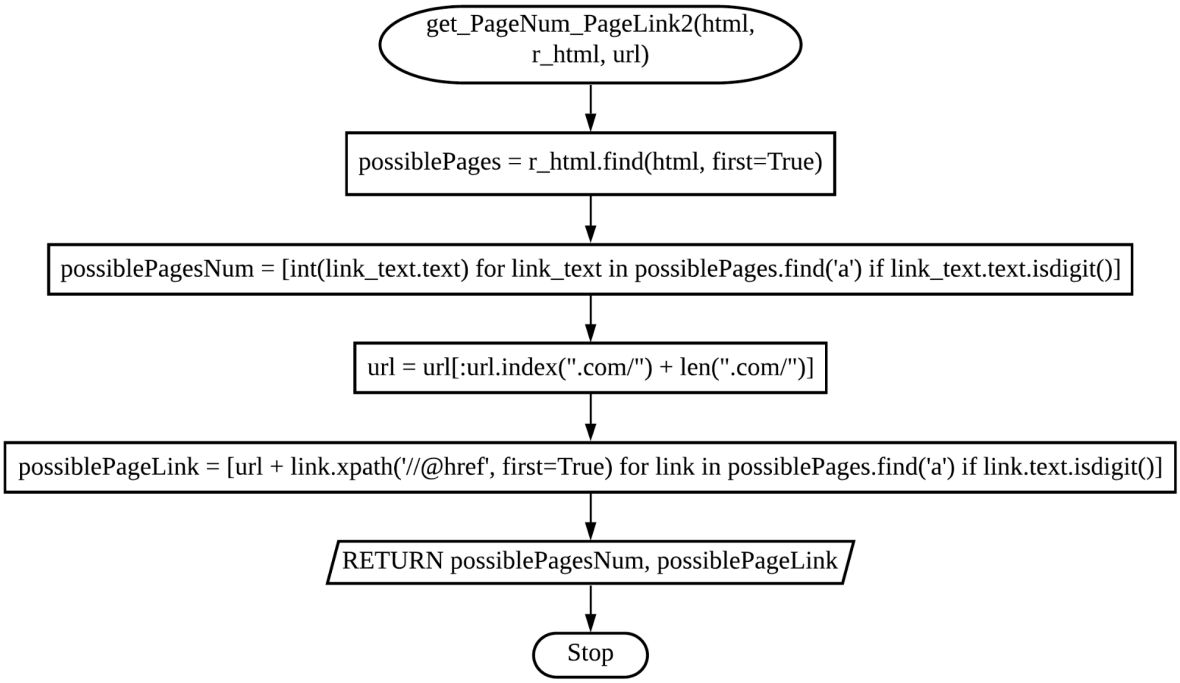


Figure 5.20: get_PageNum_PageLink2() function

4. get_PageNum_PageLink2(html, r_html, url)

Description	Crawl all the anchor text of the page numbers (e.g. 1, 2, 3, ...) as well as the URLs of the page numbers. Then, store these 2 different types of results into 2 different lists respectively and return the lists .
Method Type	static

Argument	`r_html` = request-html object of a website `url` = A web forum index's URL ot thread URL `html` = html that wraps all page numbers in an index page or a thread page
Return (`list`)	Return 2 lists, where the first list contains page numbers, and the second list contains URLs for each page number.
Example	`lastPossiblePagesNum, lastPossiblePageLink = get_PageNum_PageLink2('div.PageNav', r_html, url)`
Example of return value	`lastPossiblePagesNum = [1, 2, 3, 4]` `lastPossiblePageLink = ['http://sixcrazyminutes.com/threads/mods-adding-to-abuse-rather-than-stopping-it.113736/', 'http://sixcrazyminutes.com/threads/mods-adding-to-abuse-rather-than-stopping-it.113736/page-2', 'http://sixcrazyminutes.com/threads/mods-adding-to-abuse-rather-than-stopping-it.113736/page-3', 'http://sixcrazyminutes.com/threads/mods-adding-to-abuse-rather-than-stopping-it.113736/page-4']`

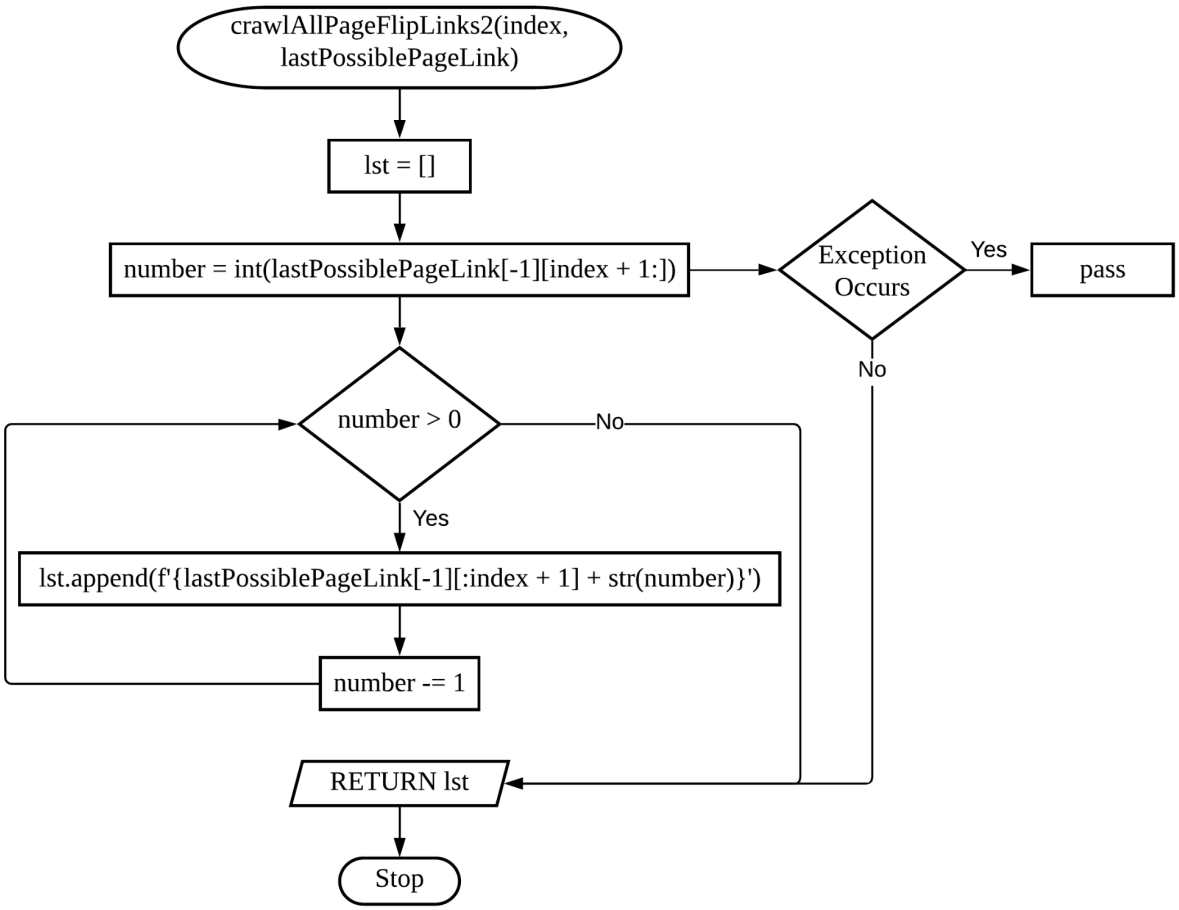


Figure 5.21: crawlAllPageFlipLinks2() function

5. crawlAllPageFlipLinks2(index, lastPossiblePageLink)

Description	This function will use the index value to get the largest page number and then iterate over the largest page number until the page number reaches 0. Within the loop, the function will append the url to a list from the largest page number to the smallest page number. Finally, return the list where it contains all the page-flipping URLs of a an index page or a thread page.
Method Type	static
Argument	`index` = An integer value of the last index of `'-'` character in the `lastPossiblePageLink[-1]` list variable. `lastPossiblePageLink` = A list that contains page-flipping URLs

Return (`list`)	Return a list that contains page-flipping URLs of index or thread pages.
Example	`page_flipping_urls2 += crawlAllPageFlipLinks2(index, lastPossiblePageLink)`
Example of return value	`['http://sixcrazyminutes.com/threads/username-missing-or-changes-needed.16626/page-2', 'http://sixcrazyminutes.com/threads/username-missing-or-changes-needed.16626/page-1', ...]`

5.4.4 ITF_Learning.py

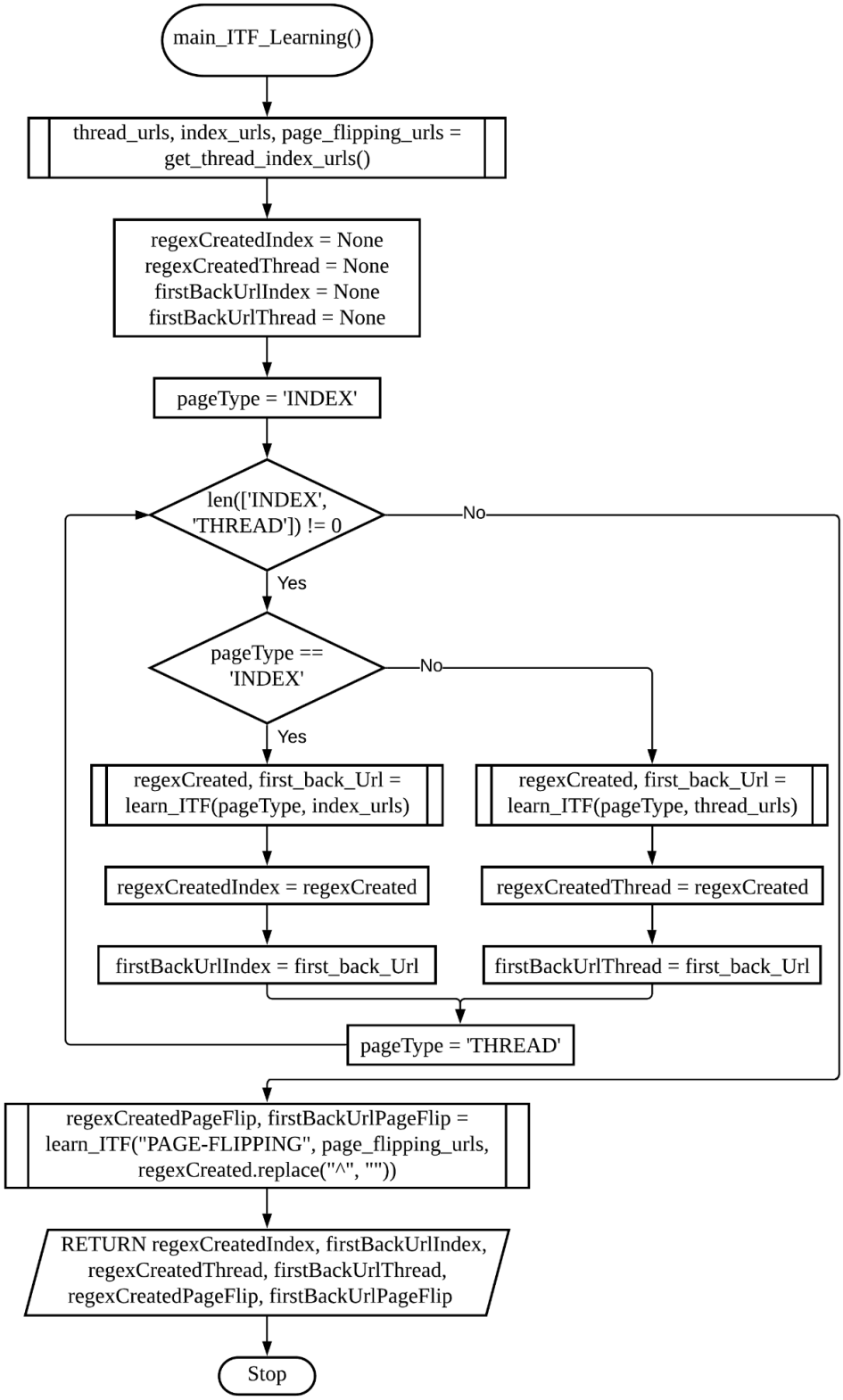


Figure 5.22: Flowchart of Index_Thread.py

5.4.4.1 Functions used in Index_Thread.py

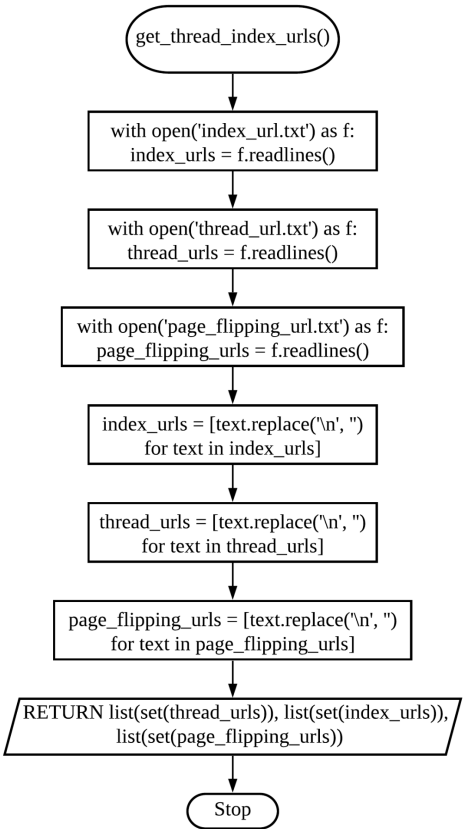


Figure 5.23: get_thread_index_urls() function

1. get_thread_index_urls()

Description	Retrieve all the links stored in *index_url.txt*, *thread_url.txt* and *page_flipping_url.txt* text files by returning each of them in a list.
Method Type	static
Argument	Do not require argument
Return (`list`)	Return 3 lists where first list contains thread URLs, second list contains index URLs and last list contains page-flipping URLs
Example	`thread_urls, index_urls, page_flipping_urls = get_thread_index_urls()`
Example of return value	`['http://sixcrazyminutes.com/threads/username-missing-or-changes-needed.16626/', 'http://sixcrazyminutes.com/threads/new-feature-nsfw-options.29244/', ...]`

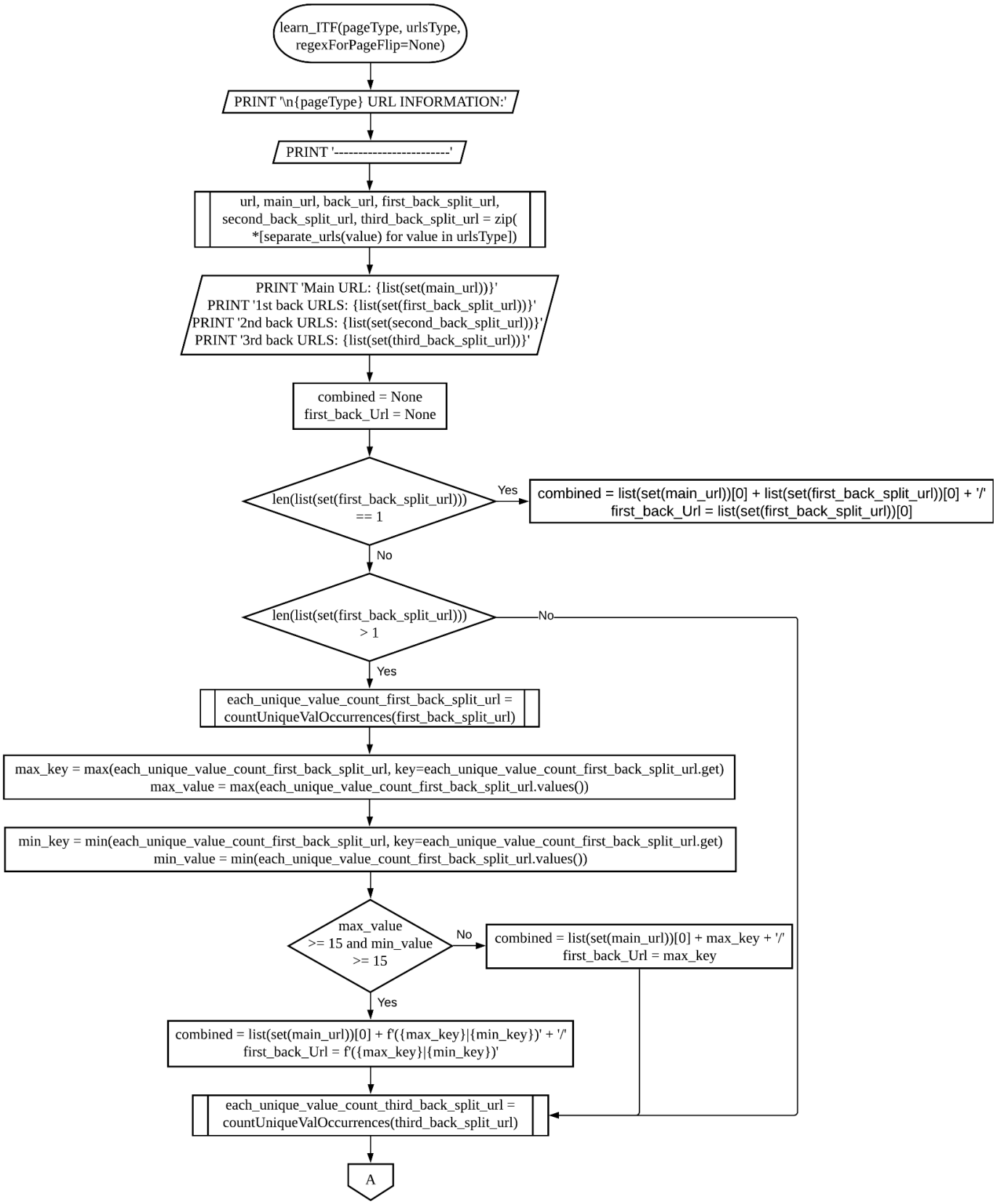


Figure 5.24: learn_ITF() function

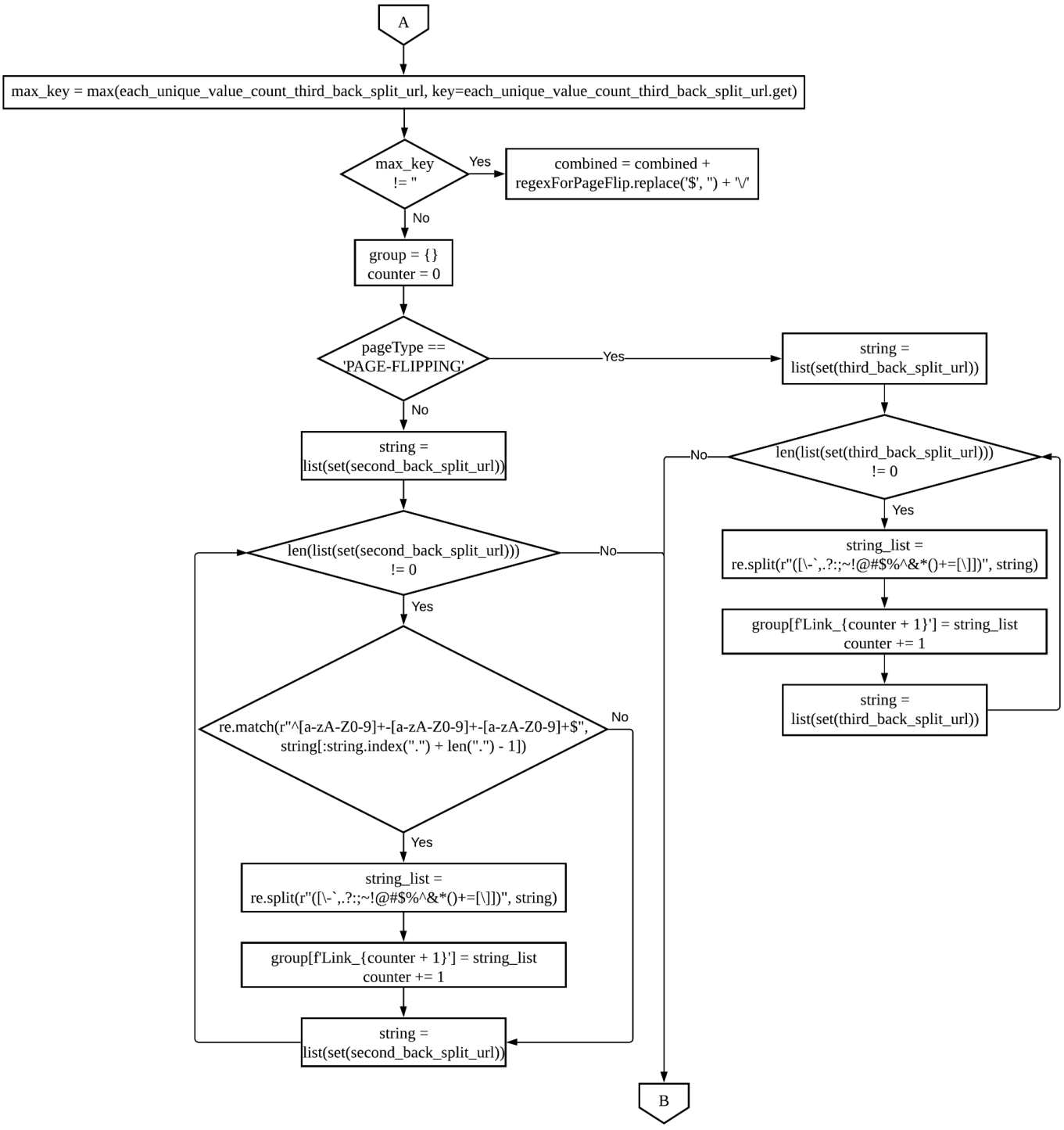


Figure 5.24: learn_ITF() function (Continue)

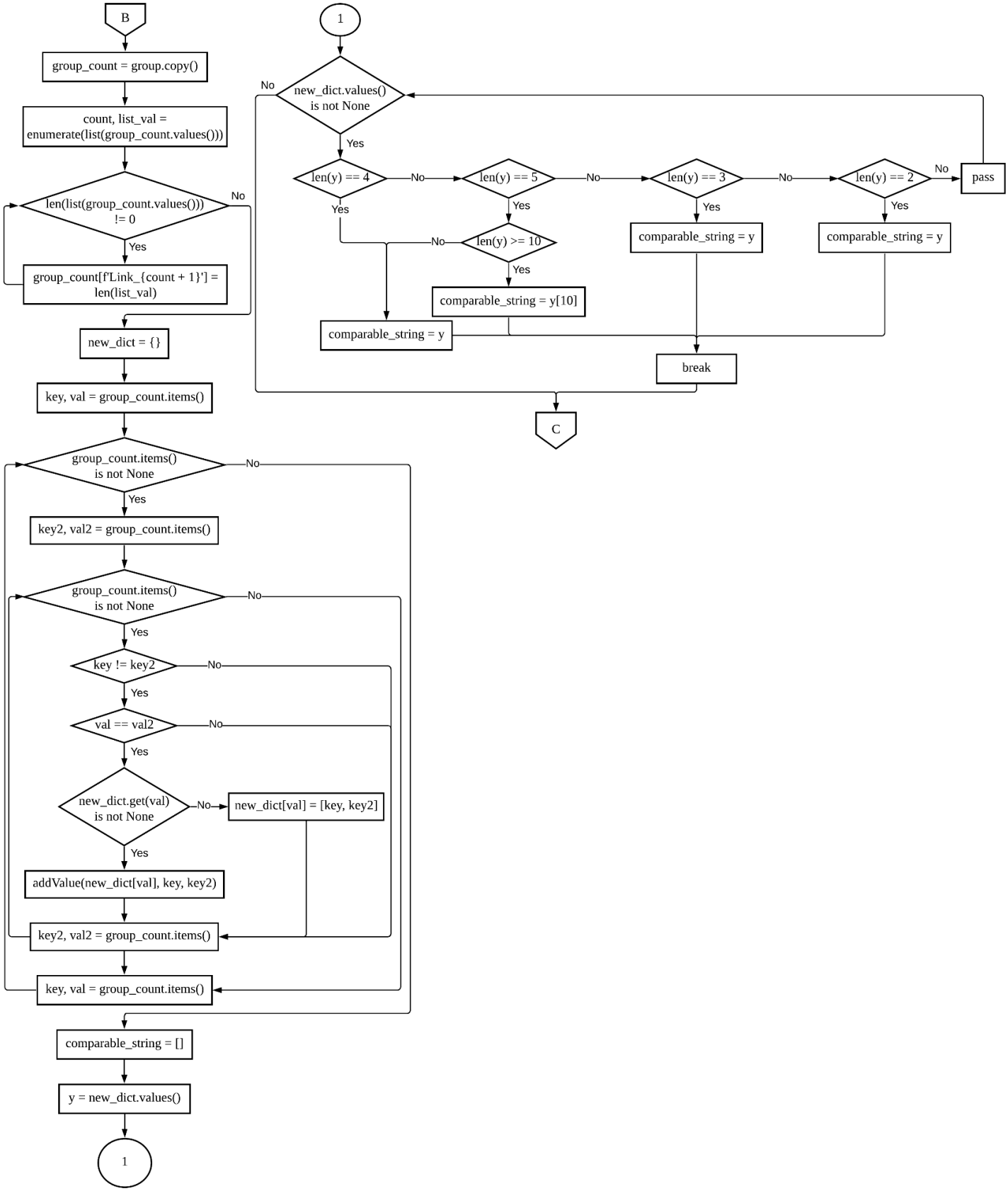


Figure 5.25: learn_ITF() function (Continue)

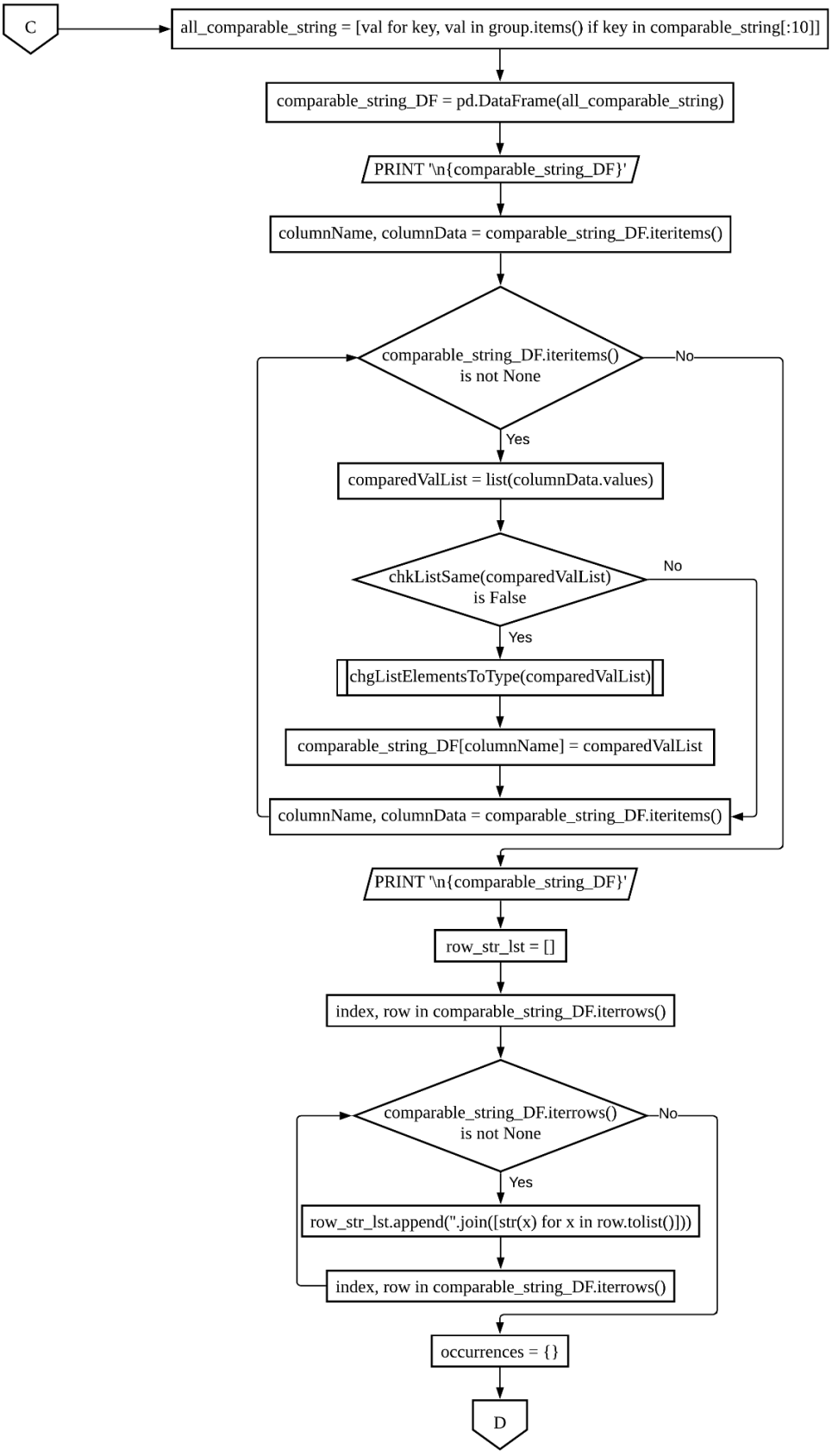


Figure 5.26: learn_ITF() function (Continue)

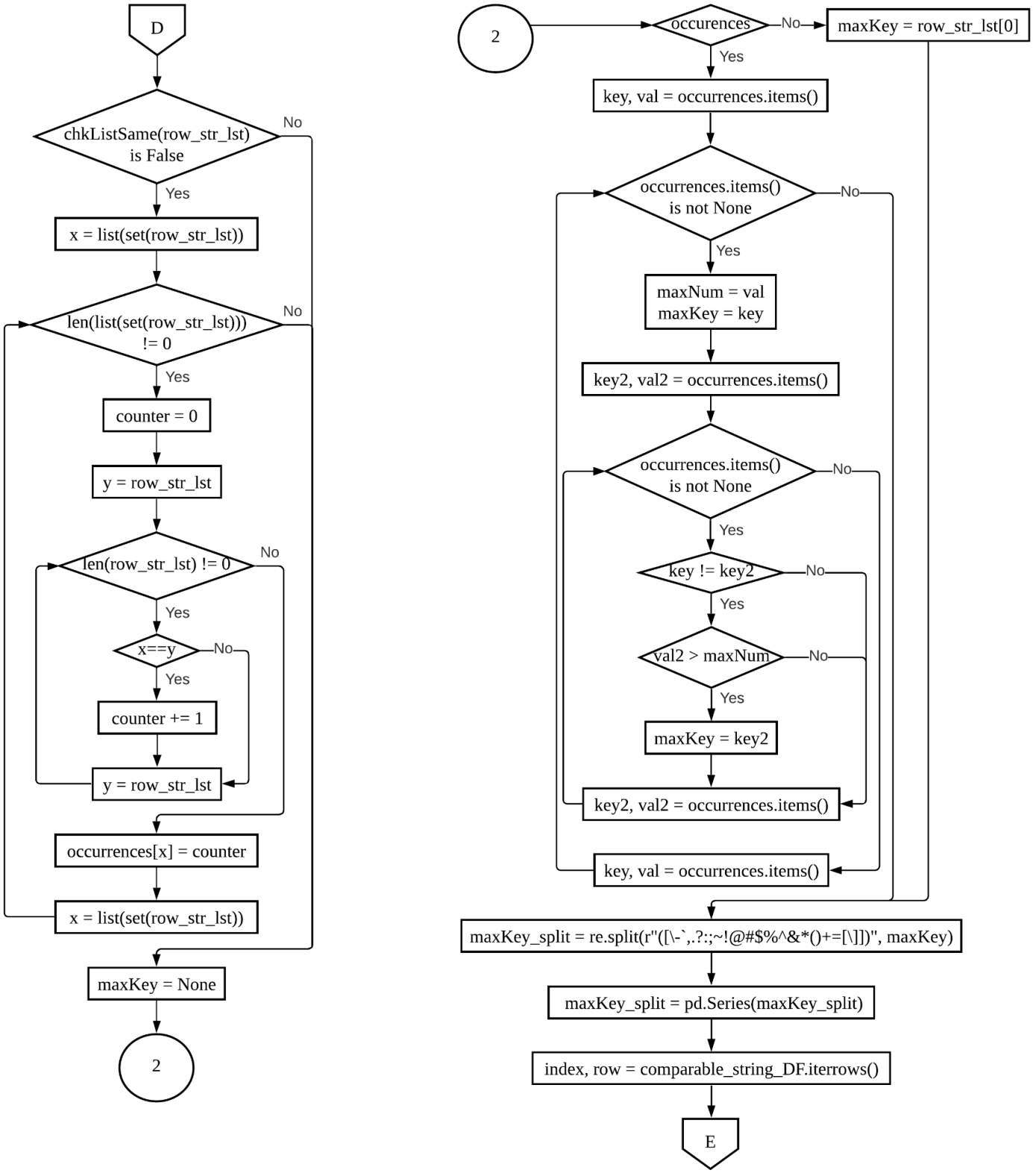


Figure 5.27: learn_ITF() function (Continue)

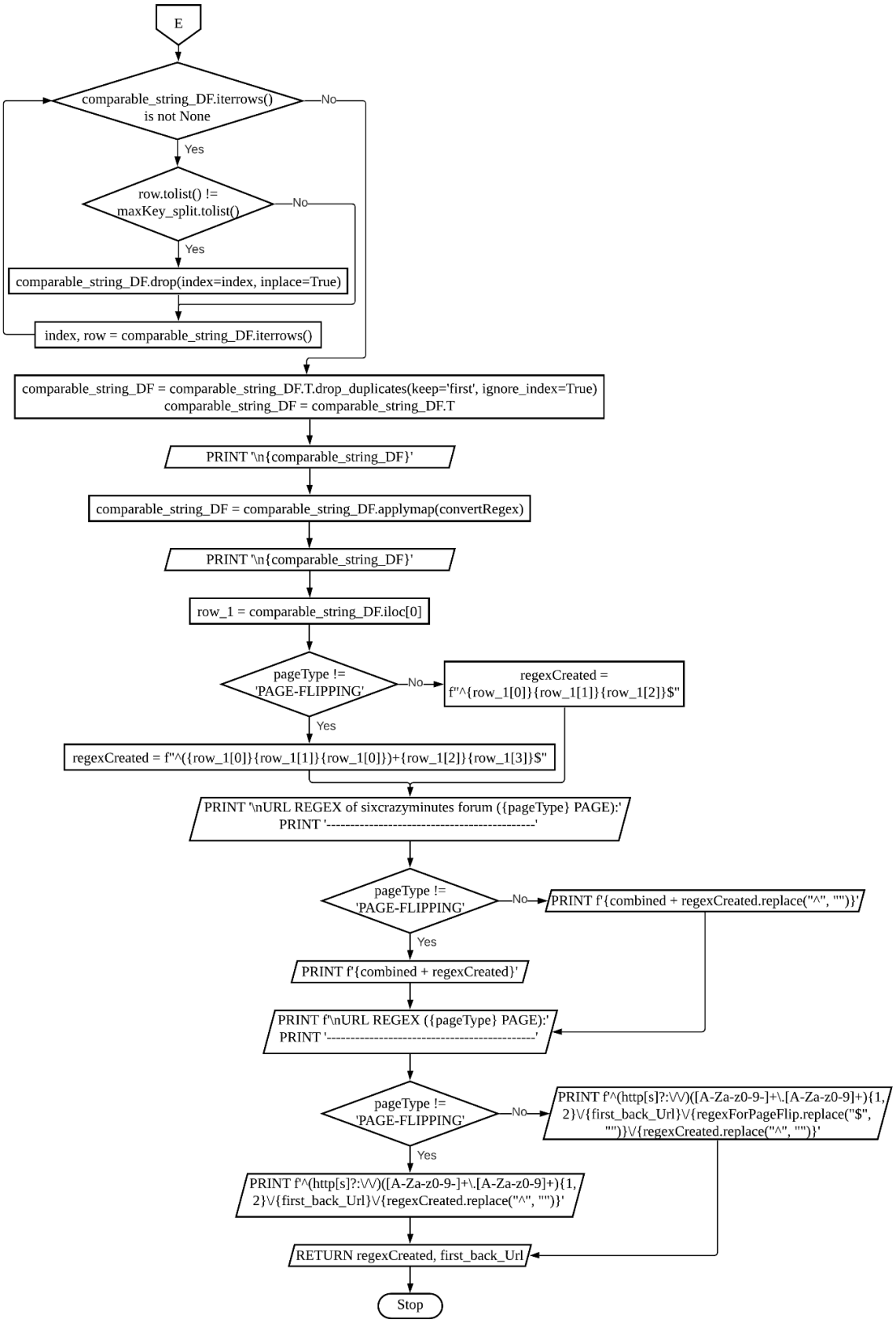
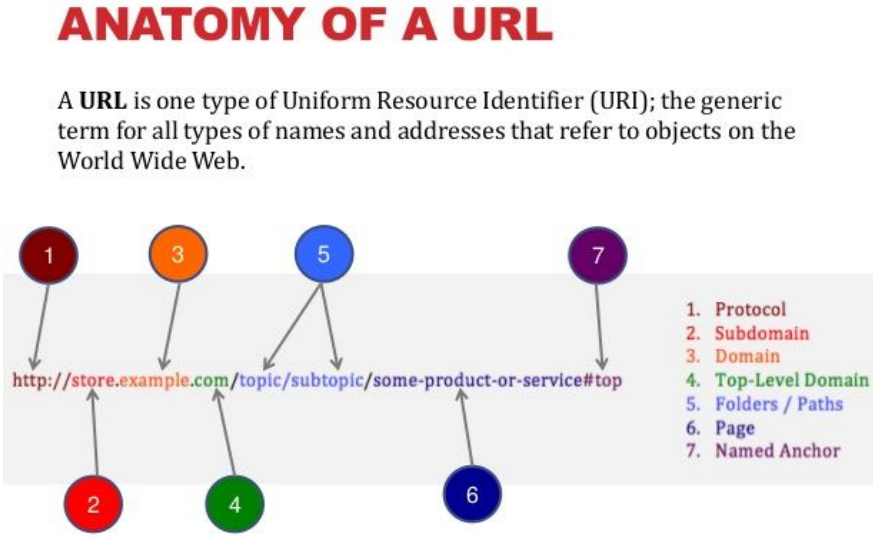


Figure 5.28: learn_ITF() function (Continue)

2. learn_ITF(pageType, urlsType, regexForPageFlip=None)

Description	This function is able to learn the link structure of index, thread and page-flipping and then create regular expressions for these 3 different types of URLs. The type of link structure to be learned is based on the argument passed in by users. Finally, it will return the regular expression learned in string type.
Method Type	static
Argument	`pageType` = The page type of the URLs to learn. E.g., Index, thread or page-flipping page. `urlsType` = The URL type to learn.E.g., Index, thread or page-flipping URLs. `regexForPageFlip` = The **page** regular expression of index or thread URL, default is `None`.
Return (`str`)	Return 2 strings where these 2 strings are regular expressions that represent **page** and **path** of an index, thread or page-flipping page respectively. For your references: 
Example	`regexCreatedPageFlip, firstBackUrlPageFlip = learn_ITF("PAGE-FLIPPING", page_flipping_urls, regexCreated.replace("^", ""))`

Example of return value	**INDEX:** `^([a-zA-Z0-9]+-[a-zA-Z0-9]+)+\.\d+$, forums` **THREAD:** `^([a-zA-Z0-9]+-[a-zA-Z0-9]+)+\.\d+$, threads` **PAGE-FLIP:** `^page-\d+$, (forums|threads)`

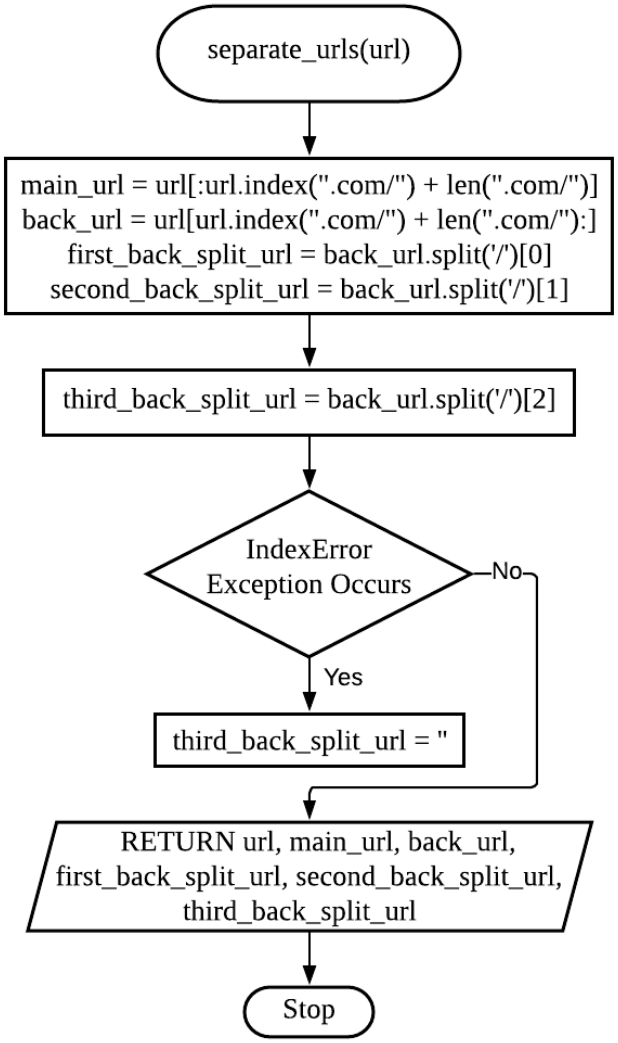


Figure 5.29: separate_urls() function

3. separate_urls(url)

Description	This function will split the URL string into `url`, `main_url`, `back_url`, `first_back_split_url`,

	`second_back_split_url` and `third_back_split_url` and then return each of them in tuple type. Example is shown as below:
	`URL: ('http://sixcrazyminutes.com/forums/the-football-forum.4/',)`
	`MAIN URL: ('http://sixcrazyminutes.com/',)`
	`BACK URL: ('forums/the-football-forum.4/',)`
	`FIRST BACK SPLIT URL: ('forums',)`
	`SECOND BACK SPLIT URL: ('the-football-forum.4',)`
	`THIRD BACK SPLIT URL: ('',)`

Method Type	static
Argument	`url` = An index, thread or page-flipping link.

Return (`tuple`)	Return 6 tuples which are `url`, `main_url`, `back_url`, `first_back_split_url`, `second_back_split_url` and `third_back_split_url`. Example is shown as below:
	`URL: ('http://sixcrazyminutes.com/forums/the-football-forum.4/',)`
	`MAIN URL: ('http://sixcrazyminutes.com/',)`
	`BACK URL: ('forums/the-football-forum.4/',)`
	`FIRST BACK SPLIT URL: ('forums',)`
	`SECOND BACK SPLIT URL: ('the-football-forum.4',)`
	`THIRD BACK SPLIT URL: ('',)`

Example	`url, main_url, back_url, first_back_split_url, second_back_split_url, third_back_split_url = zip(*[separate_urls(value) for value in urlsType])`

Example of return value	`URL: ('http://sixcrazyminutes.com/forums/the-football-forum.4/',)`
	`MAIN URL: ('http://sixcrazyminutes.com/',)`
	`BACK URL: ('forums/the-football-forum.4/',)`
	`FIRST BACK SPLIT URL: ('forums',)`
	`SECOND BACK SPLIT URL: ('the-football-forum.4',)`
	`THIRD BACK SPLIT URL: ('',)`

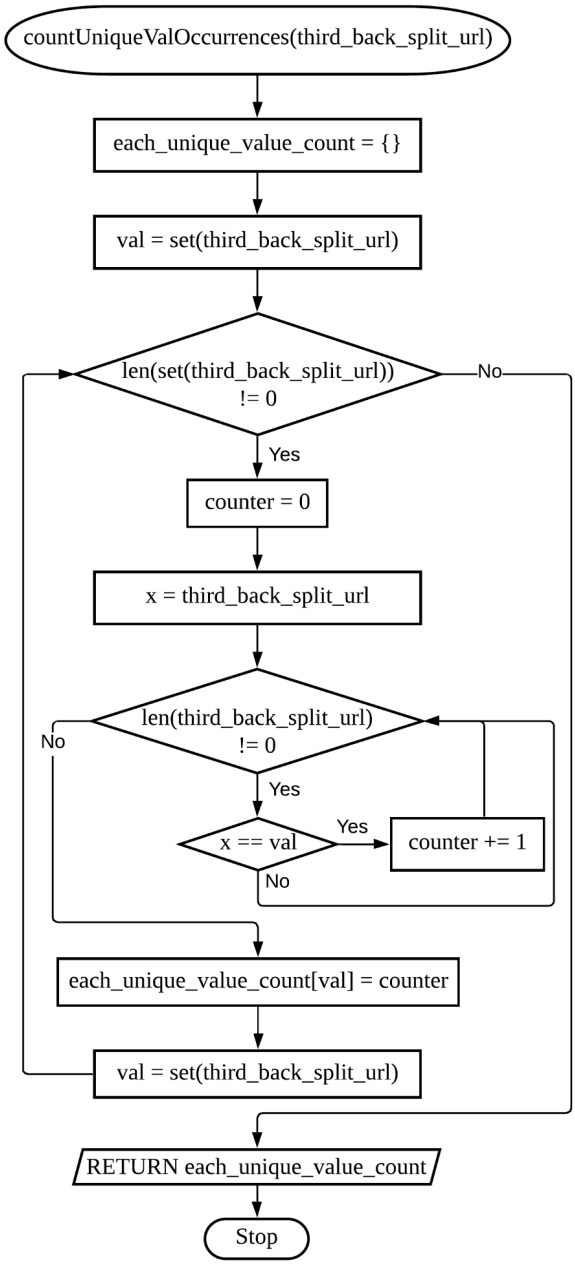


Figure 5.30: countUniqueValOccurrences() function

4. countUniqueValOccurrences(third_back_split_url)

Description	Find the unique values (Keys) in a list or tuple and then calculate the number of occurrences (Values) of each unique value just found. Finally, store them as a dictionary and return it where **unique values of the passed in list** are the **keys** while the **number of occurrences of each key** is **values**.

Method Type	static
Argument	`third_back_split_url` = A list that contains `first_back_split_url` or `third_back_split_url`.
Return (`dict`)	Return a dictionary that contains unique values in a list or tuple as keys and their number of occurrences as values.
Example	`each_unique_value_count_first_back_split_url = countUniqueValOccurrences(first_back_split_url)`
Example of return value	If the passed in tuple `first_back_split_url` = ('threads', 'threads', 'forums') Output of this function would be: `{'forums': 1, 'threads': 2}`

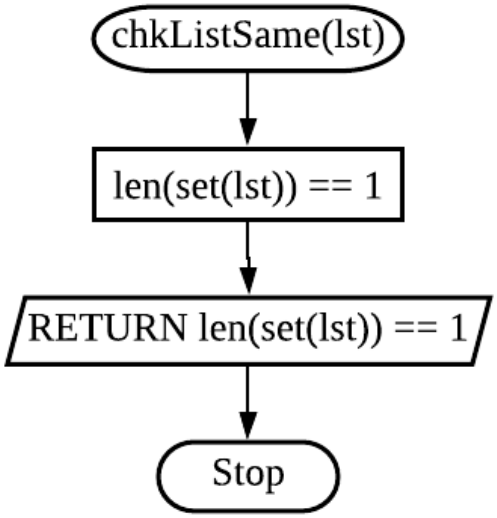


Figure 5.31: chkListSame() function

5. chkListSame(lst)

Description	Return **True** if all the elements in a list are the same. Otherwise, **False**
Method Type	static
Argument	`lst` = A list variable.
Return (`bool`)	Return **True** if all the elements in a list are the same. Otherwise, **False**
Example	`if chkListSame(row_str_lst) is False:`

Example of return value	True

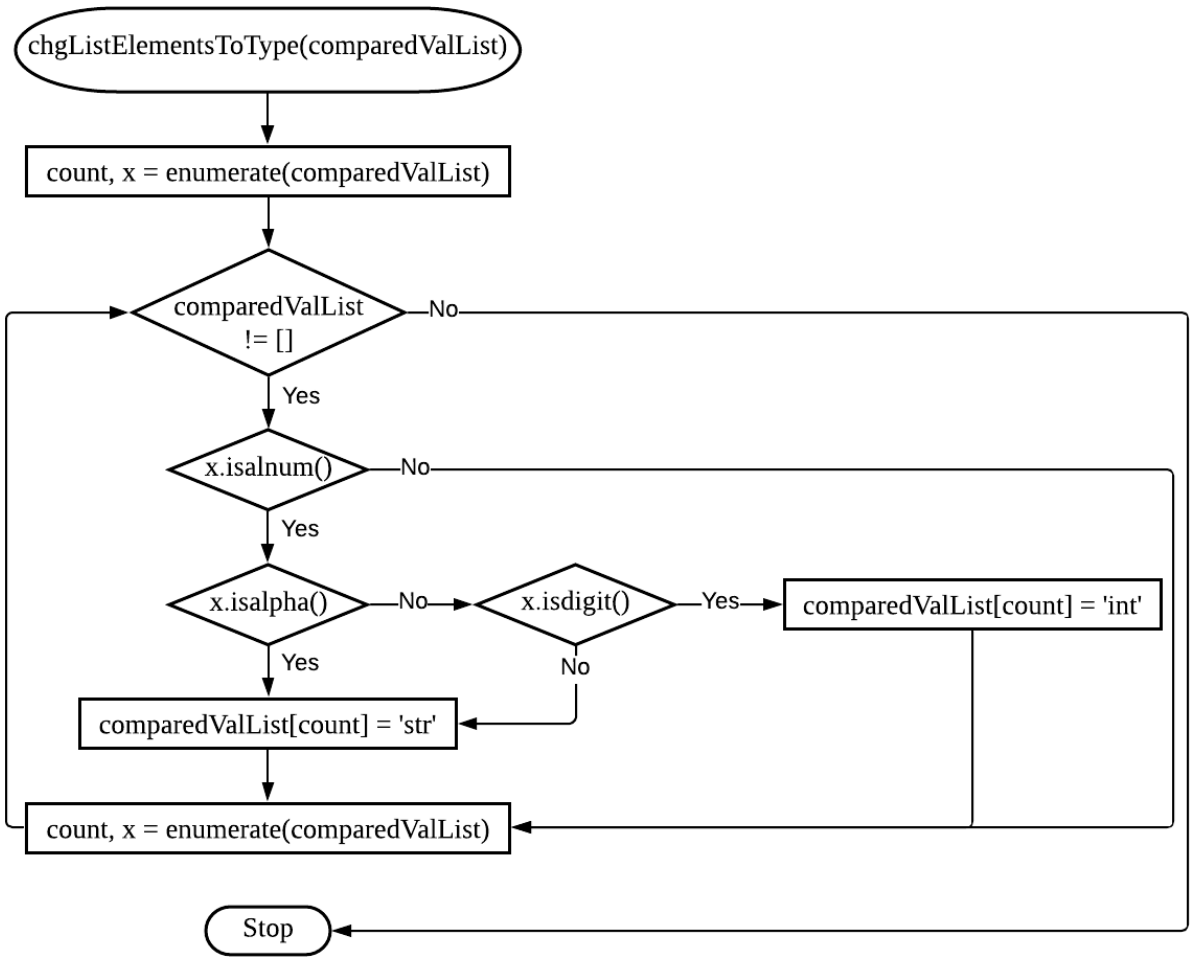


Figure 5.32: chgListElementsToType() function

6. chgListElementsToType(comparedValList)

Description	This function will change each element in a list to its respective data type If the element is a number, no matter if it is in string type or int type, this function will convert it to int. For example, if there is a list containing [‘hello’, 4, ‘5’] , the output would be [‘str’, ‘int’, ‘int’].
Method Type	static
Argument	`comparedValList` = A list variable.
Return	Do not return anything
Example	`chgListElementsToType(comparedValList)`

Example of return value	Do not return anything

5.4.5 ForumCrawler.py

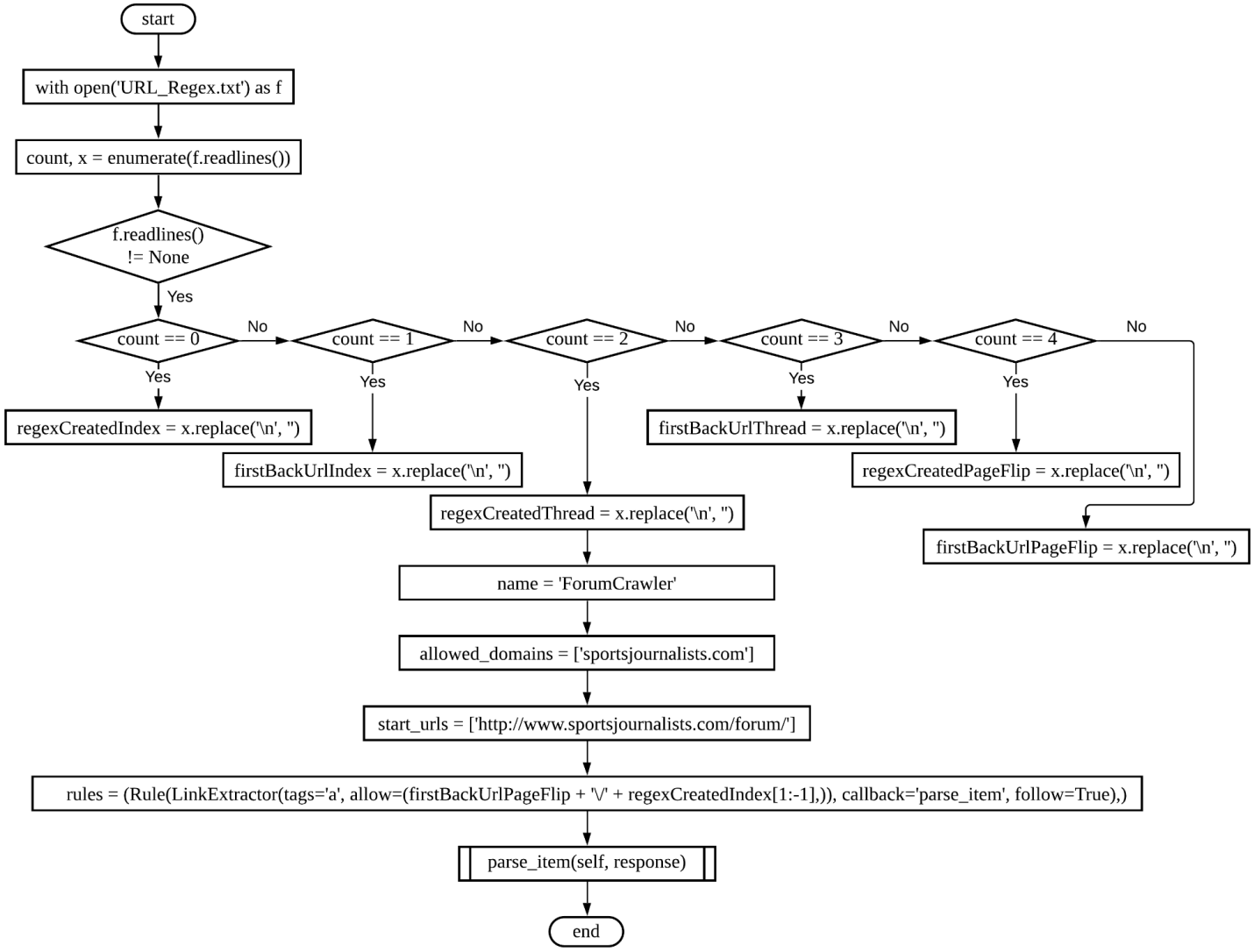


Figure 5.33: Flowchart of ForumCrawler.py

5.4.5.1 Functions used in ForumCrawler.py

Figure 5.34: parse_item() function

1. parse_item(self, response)

Description	This function will filter the extracted URLs that do not match the regular expression of index, thread and page-flipping URL. Then, this function will append those filtered URLs into 3 different list variables where `pageFlipUrls` stores page-flipping URLs, `indexUrls` stores index URLs and `threadUrls` stores thread URLs. Finally, save all these URLs into a csv file with well-labeled each URL type whether it is index, thread or page-flipping.
Method Type	Instance
Argument	`response` = It is an instance of `TextResponse` that holds the page content and has further helpful methods to handle it.
Return	Do not return anything
Example	`rules = (Rule(LinkExtractor(tags='a', allow=(firstBackUrlPageFlip + '\/' + regexCreatedIndex[1:-1],)), callback='parse_item', follow=True),)`
Example of return value	Do not return anything

5.5 Execution Session

In this section, several web forums have been selected for multiple activities experiments under 2 different conditions. One is crawling during the ITF regex learning, another one is crawling using the ITF regex learned. The purpose is to compare the differences between the number of URLs extracted during the ITF regex learning and using the ITF regex learned in which I will compare the activities of **Index URLs Extraction**, **Thread URLs Extraction** and **Page-flipping URLs Extraction**. The number of URLs crawled for each activity of each web forum is recorded for.

Activities/Forums	**Experiment 1**	**Experiment 2**	**Experiment 3**
	SixCrazyMinutes	UK of Equestria	GardenStew
Index URLs Extraction	9	37	44
Thread URLs Extraction	23	20	21
Page-flipping URLs Extraction	2362	783	1871

*Table 5.5.1: Crawling results **during** the ITF regex learning*

Activities/Forums	**Experiment 1**	**Experiment 2**	**Experiment 3**
	SixCrazyMinutes	UK of Equestria	GardenStew
Index URLs Extraction	45	81	106
Thread URLs Extraction	744	703	3381
Page-flipping URLs Extraction	2527	4203	1787

*Table 5.5.2: Crawling results **using** the ITF regex learned (10 minutes)*

Note: Crawling using the ITF regexes that have been well trained or learned will keep crawling the wanted URLs from the web forum until all the links in the web forum are gone through. This will lead to a long duration while crawling, can be up to more than 6 hours as there are many links within a web forum, the crawled results also will be very large indirectly. Therefore, I will just use 10 minutes for the ITF regexes learned to crawl the web forums in this section and compare the results.

Step 1: Index URLs Extraction

As mentioned earlier, the system will first accept the homepage url from a web forum to act as the entry page for crawling and regex learning. So, after the homepage url of a web forum is feeded into the system, the system will start crawling index URLs as much as possible that present in this forum. The crawled results will then be stored in a text file, namely *index_url.txt* once the activity of index url crawling is finished. From the tables above, we can see that the index URLs crawled are relatively lesser during the ITF learning stage as compared to the crawling stage using the ITF regexes (ForumCrawler.py) that have been well trained and learned.

Step 2: Thread URLs Extraction

After the index URLs crawling is done, the system will continue to crawl thread URLs in the web forum as much as possible where the system will retrieve all the links inside the *index_url.txt* which have crawled and saved just now in step 1 to extract all the possible thread URLs exist in those links. The crawled results will then be stored in a text file, namely *thread_url.txt* once the activity of thread url crawling is finished. From the tables above, we can see that the thread URLs crawled are relatively lesser during the ITF learning stage as compared to the crawling stage using the ITF regexes (ForumCrawler.py) that have been well trained and learned.

Step 3: Page-flipping URLs Extraction

After the threads URLs crawling is done, the system will continue to crawl page-flipping URLs in the web forum as much as possible where the system will retrieve all the links inside the *index_url.txt* and *thread_url.txt* which have crawled and saved just now in step 1

and step 2 to extract all the possible page-flipping URLs exist in those links. The crawled results will then be stored in a text file, namely *page-flipping_url.txt* once the activity of page-flipping url crawling is finished. From the tables above, we can see that the page-flipping URLs crawled are relatively lesser during the ITF learning stage as compared to the crawling stage using the ITF regexes that (ForumCrawler.py) have been well trained and learned.

5.6 Evaluation and Discussion

As shown in Section 5.5, the 3 web forums achieved a better result when utilizing the learned regexes (known as **FoCUS Crawler)** to crawl the index, thread and index URLs. With a duration of just 10 minutes, this FoCUS crawler can perform much better than normal crawling that utilizes DOM tree locating approach such as locating HTML tags using selenium, beautifulsoup4 and so on to crawl the required links in a website, where this crawler can crawl relatively more links for each activity in every web forum.

As stated earlier, given a forum, FoCUS first learns a set of ITF regexes and then it performs online crawling using a breadth-first strategy. This makes FoCUS efficient in online crawling, especially in large scale crawling because it only needs to apply the learned ITF regexes on the newly come-in URLs. It will continue crawling the URLs from the web forum until all the links in the web forum are gone through and it will match each link or URL with the learned ITF regexes during this process. FoCUS does not need to group outgoing URLs, classify pages, detect page-flipping URLs, or learn regexes again for that forum. Such time consuming operations are only performed during its learning phase. Meaning that the FoCUS crawler just needs to learn once and crawl multiple times. This can be proven by referring to **Section 5.5**, I use the homepage URL of SixCrazyMinutes forum to act as the entry page during the learning phase (refer to *Table 5.5.1*). After learning the ITF regexes, I just need to pass the homepage URL of SixCrazyMinutes forum to the FoCUS crawler to let the crawler to automatically crawl all the links I want (refer to *Table 5.5.2*). This time, the FoCUS Crawler can crawl a large amount of correct links to the user.

Besides, another good thing about FoCUS is that it can crawl other web forums as well if other web forums' link structures are similar to the one I learned. That's why it is good for large scale online crawling. For example, the ITF regexes learned can be applied to crawl the links in UK of Equestria and GardenStew forums as well even though I use the SixCrazyMinutes forum in the learning phase. This is because the UK of Equestria and GardenStew forums are having the similar link structure as SixCrazyMinutes forum so that the ITF regexes can be applied on these

2 forums. In this paper, I have tried to use ITF regexes learned based on the SixCrazyMinutes forum to crawl on another 9 web forums which having the similar link structure and forum software package as SixCrazyMinutes forum. As expected, the FoCUS crawler is able to crawl all the other 9 web forums index, thread and page-flipping URLs or links effectively. This proves the efficiency and convenience of FoCUS where it achieves the saying of learn once and crawl multiple times. The details of the 9 web forums that I have tried out are available on my GitHub. Through the experiment, I believe that the FoCUS is able to crawl most of the web forums that are having similar link structure and built by similar forum software packages as the web forum that you use in the learning phase.

5.7 Highlight of Contribution and Conclusion

In this paper, I proposed and implemented FoCUS, a supervised forum crawler that is used to crawl the index, thread and page-flipping URLs in the web forums. I reduced the forum crawling problem to a URL type recognition problem and designed methods to learn ITF regexes. Then, utilize the ITF regexes to crawl on other web forums by matching all links or URLs in every other forum to the ITF regexes learned. The experiment in this paper mainly uses three web forums for analysis and evaluation which include SixCrazyMinutes, UK of Equestria and GardenStew web forums. Besides, the overall workflow is elucidated in this paper as well as several diagrams are provided for better understanding such as Layered Diagram, System Block Diagram, Data Flow Diagram and Flowcharts. Instead of locating DOM trees like locating HTML tags to crawl all the required links in a web forum, the FoCUS approach can effectively crawl all the required links in different web forums with minimal efforts.