Task 4: CrawlerAnalysis

Features

- **Access** a website,...
- **Extract** content from a website,...
- **Publish** content on a website,...

all **automatically**.

Downloads

Crawler analysis

How to Use

To run the CrawlerAnalysis script requires the third-party library selenium which can be installed using pip.

```
pip install selenium
```

To run selenium on ubuntu in combination with firefox requires to uninstall the version provided by snap and use the personal package archive by the Mozilla Team. The process is described here.

Running the CrawlerAnalysis script is simple. After initializing the environment variables (lines 1 and 2), the script can be executed (line 3). Note, that on ubuntu the history command allows by default to display the last 1000 commands entered. To keep the password private, the entry number <n> of the history should be inferred (line 4) and deleted (line 5).

```
jan@linux:~$ export CHECK_LOGIN_USR="jan.lotze@bearingpoint.com"
jan@linux:~$ export CHECK_LOGIN_PWD="************"
jan@linux:~$ history | less
jan@linux:~$ history -d <n>
jan@linux:~$ python3 crawler_analysis.py
```

Dependencies

Python Standard Library:

- os

Third-party library:

- selenium

Technical Summary

After the credentials of the user are inferred, a webdriver is used to login to the forum. From there, a text is extracted, which is analyzed. Afterwards, the analysis is submitted to the forum.

Tasks & Design choices

Main function

First, the credentials of the user are extracted from the prepared environment variables. Next, the browser/driver together with a waiting-time is set up. Then the forum website is opened. Next, several login procedures (pre-stored data, Microsoft, Bearingpoint) are passed and the question, whether one should stay signed in, is answered. This is, where the login credentials are required. Once logged in, the text on the forum website is downloaded, analyzed and stored as a string. FInally, the analysis is submitted to the forum.

```
def main():
    """Login to forum, read text, analyze it and submit the analysis."""

    user_name, user_passwd = get_credentials()

    browser,wait = setup_browser()

    browser.get("https://learnhub.bearingpoint.com/mod/forum/view.php?id=1808")

    browser = login_forum(browser,wait,user_name,user_passwd)

    # forum text
    element = wait.until(
    EC.presence_of_element_located((By.XPATH, \
                           "//*[@id='intro']"))
    )
    forum_text = element.text
    forum_url = browser.current_url

    analysis = text_analysis(browser,wait,forum_text)

    submit_analysis(browser,wait,forum_url,analysis)

    browser.close()
```

Preparation

- **Task**: Login to the forum.
    - **Implementation**: The login credentials are obtained from previously prepared environment variables.

    ```
    def get_credentials():
        """Get login credentials from environment variables.."""

        user_name = os.getenv('CHECK_LOGIN_USR')
        user_passwd = os.getenv('CHECK_LOGIN_PWD')

        return user_name,user_passwd
    ```

    - **Q**: Why are environment variables used?
      A: While environment variables are not safe since they can be echoed, they are restricted to the shell session and can be prepared in private.
- **Task**: Prepare the webdriver and a waiting-time in case the websites or elements thereon do not reply immediately.
    - **Implementation**: Using the *webdriver.Firefox* and *WebDriverWait* class, a webdriver and wait object are created. The former resembles the current website and allows access to elements on it, while the latter provides a delaytime for the websites and their elements to respond.

    ```
    def setup_browser():
        """Setup browser."""

        browser = webdriver.Firefox()
        wait = WebDriverWait(browser, 10)

        return browser,wait
    ```

Login(s)

- **Task**: Login to the forum to add an entry.

- **Implementation**: In all login steps, the XPATH language is used to identify elements on the website based on attributes. These are chosen such, that they are reasonably connected to the purpose of the element.

  First, the link characterized by @href containing the word "login" is clicked. Then, during the SAML login, the link containing "/auth/saml2" are used. For the Microsoft login, the E-Mail address (previously calld username) is requested. Hence the input-element with type "email" is used. To continue, the input-element with value "Next" is clicked. The Bearingpoint login requires a password which is entered in the passwordInput. To continue, the password is submitted via the submitButton. Last, to answer the question whether one should stay logged in, the "No" button is pressed.

  ```
  def login_forum(browser,wait,user_name,user_passwd):
      """Cascade of logins."""

      element = browser.find_element(By.XPATH, \
                             "//*[contains(@href,'login')]")
      element.click()

      # SAML login
      element = wait.until(
      EC.element_to_be_clickable((By.XPATH, \
                       "//*[contains(@href,'/auth/saml2')]"))
      )
      element.click()

      # MS login
      element = wait.until(
      EC.presence_of_element_located((By.XPATH, \
                             "//input[@type='email']"))
      )
      element.click()
      element.clear()
      element.send_keys(user_name)
      element = wait.until(
      EC.element_to_be_clickable((By.XPATH, \
                       "//input[@value='Next']"))
      )
      element.click()

      # BP login
      element = wait.until(
      EC.presence_of_element_located((By.XPATH, \
                             "//input[@id='passwordInput']"))
      )
      element.click()
      element.clear()
      element.send_keys(user_passwd)
      element = wait.until(
      EC.element_to_be_clickable((By.XPATH, \
                       "//*[@id='submitButton']"))
      )
      element.click()

      # Stay signed in?
      element = wait.until(
      EC.element_to_be_clickable((By.XPATH, \
                       "//input[@value='No']"))
      )
      element.click()

      return browser
  ```

- **Q**: How are the attributes used to identify the elements found?
  A: Pressing F12 in Firefox opens the Developer Tools application. Splitting it off into its own window and putting it in Inspector-mode next to the browser window, the code elements can be identified with the website elements. Expanding the entries an diving down the tree of nested expressions allows to find the line of code yielding the action or event one is interested in. Using the Console-mode of the Developer Tools allows to determine a pattern to match for these entries. Conceptually it is similar to regex, but syntactically it is closer to the XPATH language.

Analysis

- **Task**: Analyze the text extracted from the forum.
    - **Implementation**: The analysis is simple and consists of three parts. First, the fraction of unique words, irrespective of capitalization, is determiend. Next, a histogram of word length is constructed. Last, the number of occurences of the word "und" is counted and it is shown as * in the histogram. Additional information, such as quotes of all capitalized words from zitate.de failed at a relentless popup window which seemingly stayed out of reach of the webdriver.

```
def text_analysis(browser,wait,forum_text):
    """Perform the text analysis."""

    # split text into list of words
    word_list = forum_text.split()


    # unique words
    unique_words = set(map(str.lower,word_list))

    ratio = f"Bruchteil einmalig vorkommender Wörter:\n"+ \
             "{len(unique_words)}/{len(word_list)}"

    # special word
    special_word = "und"
    special_length = len(special_word)
    special_number = len([word for word in word_list \
                          if word==special_word])

    # histogram of word-lengths
    lengths = [len(word) for word in word_list]
    # count lengths
    dct = {length:0 for length in range(max(lengths)+1)}

    for length in lengths:
        dct[length] += 1
    # sort lengths
    lengths_sort = list(dct.keys())
    lengths_sort.sort()
    output_lst = ["Histogramm der Wortlängen. "+ \
                  "Das Wort \"und\" ist durch * markiert:"]
    # histogram
    for length in lengths_sort:
        if (length==special_length):
            output_lst.append(str(length).zfill(2) \
                             +" | "+"*"*special_number \
                             +"#"*dct[length])
        else:
            output_lst.append(str(length).zfill(2) \
                             +" | "+"#"*dct[length])

    analysis = ratio+"\n\n"+"\n".join(output_lst)

    return analysis
```

Submission

- **Task**: Submit the analysis to the forum.
    - **Implementation**: First, the button to add a new subject is pressed. This expands a form, where the "extended" input link is clicked. Next, the subject field is inferred and filled. Then, the analysis is added to the textbox. After the tags are added one after the other, the contribution is submitted.

```
def submit_analysis(browser,wait,forum_url,analysis):
    """Write analysis to forum."""

    # overcome problems with expandable elements
    expanded = False
    while (not expanded):

        try:
            browser.get(forum_url)
            expanded = True
            element = wait.until(
            EC.element_to_be_clickable((By.XPATH, \
                             "//*[contains(text(),'Neues Thema hinzufügen')]"))
            )
            element.click()

            element = wait.until(
            EC.element_to_be_clickable((By.XPATH, \
                             "//input[@value='Erweitert']"))
            )
        except TimeoutException:
            expanded = False

    element.click()


    # enter subject
    element = wait.until(
    EC.presence_of_element_located((By.XPATH, \
                           "//input[@name='subject']"))
    )
    element.click()
    element.clear()
    element.send_keys("Textanalyse?")

    # enter textblock
    element = wait.until(
    EC.presence_of_element_located((By.XPATH, \
                           "//*[@role='textbox']"))
    )
    element.click()
    element.clear()
    element.send_keys(analysis)


    # add Tags
    # enter Selenium
    element = wait.until(
    EC.presence_of_element_located((By.XPATH, \
                   "//input[starts-with(@placeholder,'Tags')]"))
    )
    element.click()
    element.clear()
    element.send_keys("Selenium,")
    # click somewhere else
    element = browser.find_element(By.XPATH, \
                           "//input[@name='subject']")
    element.click()

    # enter Python
    element = wait.until(
    EC.presence_of_element_located((By.XPATH, \
                           "//input[starts-with(@placeholder,'Tags')]"))
    )
    element.click()
    element.clear()
    element.send_keys("Python,")
    # click somewhere else
    element = browser.find_element(By.XPATH, \
                           "//input[@name='subject']")
    element.click()

    # enter Text
    element = wait.until(
    EC.presence_of_element_located((By.XPATH, \
                           "//input[starts-with(@placeholder,'Tags')]"))
    )
    element.click()
    element.clear()
    element.send_keys("Text,")
    # click somewhere else
    element = browser.find_element(By.XPATH, \
                            "//input[@name='subject']")
    element.click()

    # enter Analyse
    element = wait.until(
    EC.presence_of_element_located((By.XPATH, \
                            "//input[starts-with(@placeholder,'Tags')]"))
    )
    element.click()
    element.clear()
    element.send_keys("Analyse,")
    # click somewhere else
    element = browser.find_element(By.XPATH, \
                            "//input[@name='subject']")
    element.click()

    # submit contribution
    element = wait.until(
    EC.presence_of_element_located((By.XPATH, \
                            "//input[@name='submitbutton']"))
    )
    element.click()
```

- **Q**: Why is the construct of an try-except statement in a while loop necessary?
  A: In my tests there were situations when the expandable would not open at random, despite being clicked. As a result, the form with the "extended" button would not be uncovered and a TimeOutException raised. The workaround to overcome this problem is to refresh the page, click the expandable again and repeat it until it opens.
- **Q**: What is the purpose of all these lines commented as "click somewhere else"?
  A: When entering tags, they are added when a comma is encountered or the user clicks out of the input box. Since the commas alone were not sufficient to add tags with the webdriver, the "click[s to] somewhere else" are added.

Planned features

- **Security**: Store the user credentials somewhere safe.
- **Quotations**: Add quotations for the important, i.e. capitalized words to the text analysis.
- **Insecure connections**: Handle insecure connections properly.
- **Popups**: Handle popups properly.