cere[:bruh]

:two buttons to rule them all

projectReport

:dev ProgyanDas

ES102-section2

cere[:bruh]

:METHODS :OVERVIEW :THE IDEAL

:THE IDEAL

Cerebruh aims to make the act of *knowing* as simple as possible.
In the twenty-first century, *forgetting* should not be the reason you don't know something:
That's why, with Cerebruh, you can access all the information on Wikipedia *and* the UrbanDictionary with the flick of a finger.
Yes, it's truly as simple as that. You highlight the word you don't know about, press F2 or F9, or just about any combination of your choice, and a succint summary of your word will pop up.

:OVERVIEW

**Cerebruh uses a number of different libraries, ranging from those that manipulate the GUI of Windows (pyautogui) to those that help in substituting general expressions from strings (re, regularexpression), to those that actually scrape data off of Wikipedia and UrbanDictionary.
Detailed analyses of each are in the pages that follow.**

:METHODS

Here's a list of all defined functions.

`copyHighlight()` `getLink()`

`getSummary()` `summarizeUD()`

`popUpWindow()`

copyHighlight()

```
# Function for returning what is highlighted by the cursor
def copyHighlight():
    pyperclip.copy("") # in case nothing is highlighted, this line prevents duplicate entries
    # copying what is highlighted
    # stackoverflow suggests using the hotkey function, but for some reason that does not work
    pyautogui.hotkey('ctrl','c')
    time.sleep(0.01) # suggested by stackoverflow user [soundstripe]: ctrl + C is fast but sometimes programs runs faster
    pyautogui.click(pyautogui.position()) # dispels the highlighted text and indicates that the program has responded
    return pyperclip.paste() # returns what has been copied as a STRING
```

copyHighlight() uses the **copy** function from the **pyperclip** library, the **hotkey** and **click** functions from the **pyautogui** library, and the **sleep** funcion from the **time** library. It copies what is highlighted and returns a string that contains the highlighted word(s).

getLink()

```
def getLink(search):
    # the get function "gets" the HTML/XML/JavaScript code from the website. "html5lib" is a parser.
    # StackExchange recommended a parser named "lxml", but I couldn't get it to work. "html5lib" is an alternative.
    soup = BeautifulSoup(get("https://en.wikipedia.org/w/index.php?search="+search).content, "lxml")
    for link in soup.findAll("a") :  # a, from a href, seems to be the tag that identifies links in websites.
        # I figured this out from the documentation, but had to refer to stackexchange to fix a few bugs
        link_href = link.get('href') # extracts the actual link from within the <a></a> tag.
        link_rank = link.get('data-serp-pos') # I noticed that all search elements in search have this numbered tag
        # This how I identify search results from other links. If there is no rank, it is not a search result.
        if (link_rank)!=None: # findAll gets ALL links, but if the rank exists, it is a search result.
            if "wiki/" in link.get('href'): # there are some wiktionary links that pop in, but I needed the wikis only.
                return(link.get('href')) # returns the top search result of the input string.
```

getLink() uses the **BeautifulSoup4**, or **bs4** library to webscrape Wikipedia Search for the closest link to the argument passed into it. *This was initially made to scrape Google. However, after I wrote the code, Google blocked my code since it was a bot. Apparently, Google bot-checks IP addresses when there are more than 8 requests or so per minute.*

popUpWindow()

```
def PopUpWindow(response,index): # The response is the YES/NO at the end of the message-box. If Yes, this shows more information
    if response == 1: # 1 is YES, in this case
        index+=1 # The index is incremented so the next paragraph comes up
        # This is essentially identical to the original YES/NO prompt, except this ends in a recursive construct that
        # terminates when the user presses NO.
        response = messagebox.askyesno("Cerebruh [Wiki] "+search, getSummary("https://en.wikipedia.org" + getLink(search + " -disambiguation"), index))
        PopUpWindow(response, index) # Recursion call
    print("") # Was having some issues with this function, but a print function seemed to fix things. Will have to look into why.
```

popUpWindow() uses recursion and **TKinter** to shift through wiki paragraphs

getSummary()

```
# Summarizing the Wikipedia entry was hard for two reasons: there are disambiguation pages, and there are reference tags
# I had to read through https://en.wikipedia.org/wiki/Wikipedia:Advanced_source_searching to make this work.
# I used the Regular Expressions library, re, for separating the reference tags from the text.
def getSummary(link, index):
    print(link) # for reference; a link in the prompt helps to debug where the code went wrong and it is harmless.
    soup = BeautifulSoup(get(link).content,"lxml") # lxml is an efficient parser, I used html5lib before.
    # StackExchange suggested using 'lxml' instead of 'html5lib', because it apparently works better.
    # In my experience, they appeared about the same, although 'lxml' appeared a little faster. Not confirmed.
    return(sub('\[\d*?\]','',soup.find_all('p')[index].text)) # This statement used the RegularExpressions library
    # Going through the statement character by character - RegularExpressions takes a general expression as a wildcard,
    # and the sub function replaces it with something else, in this case, a blank character.
    # '\[' & '\]' are escaped square brackets. '*d' describes a digit, '?' indicates any number of the digit.
```

getSummary() is at the soul of this project. It uses the **bs4** library as well as the **regularexpressions** library to form a summary of the text. A detailed explanation of the function is in the comments of the code.

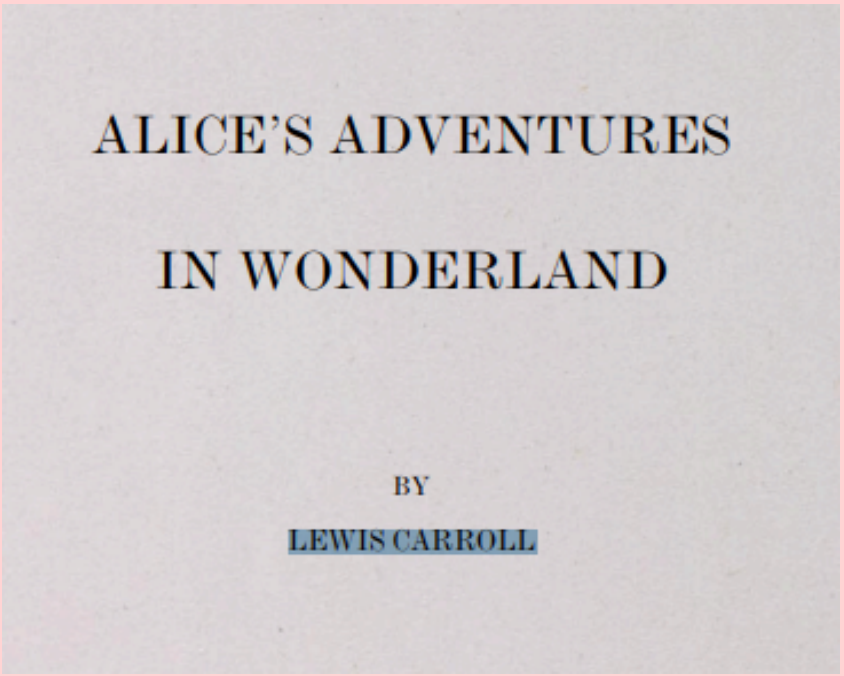
summarizeUD()

```
# Summarizing the UrbanDictionary entry was comparatively easier, but due to assignment pressure, I had to use my late days
# UrbanDictionary does not have <p> tags. Instead, it has <div> tags with a ['meaning'] attribute. The search URL is easy
# to generate, and the rest of the program follows much of the same pattern as the code above for scraping Wikipedia.
def summarizeUD(search): # Search is the returned variable from copyHighlight()
    respond = get("http://urbandictionary.com/define.php?term=" + search) # Broken into different lines for better reading
    soup = BeautifulSoup(respond.content, "lxml") # Gets the BeautifulSoup object, which contains the HTML code for the page
    string = "" # Initializing a string variable to later store the summary in it.
    returnNext = False # Some meanings don't have examples, and therefore two different meanings may pop up.
                       # this boolean prevents that from happening. Not very efficient, should improve.
    for tag in soup.find_all('div'): # In Wikipedia, I did this with the <p> tag. Here, I use the <div> tag.
        if tag.get('class') == ['meaning']: # The meaning tag identifies the definitions from useless links and other text.
            if not returnNext: # This prevents the function from returning multiple definitions.
                string += tag.text + "\n" # The '\n' is a line break, so the example is better formatted when added.
            else: # In statement above, tag.text returns the text from the <div> tag
                return(string) # If the example has already been added, this simply returns the string and does not add.
        if tag.get('class') == ['example']: # The example tag identifies an example of the word in the given meaning.
            string += "\nFor example, a sentence would be:\n" # Again, '\n' added for better formatting.
            string += tag.text # tag.text gives the text from within the <div> tag.
            return(string) # returns string if an example has been succesfully appended.
```

summarizeUD() is how **cerebruh** summarizes the UrbanDictionary. It uses a wide spectrum of **bs4** functions to arrive at a summary of the argument passed into it.

HOW DOES IT WORK?

step 1 highlight what you want to know.



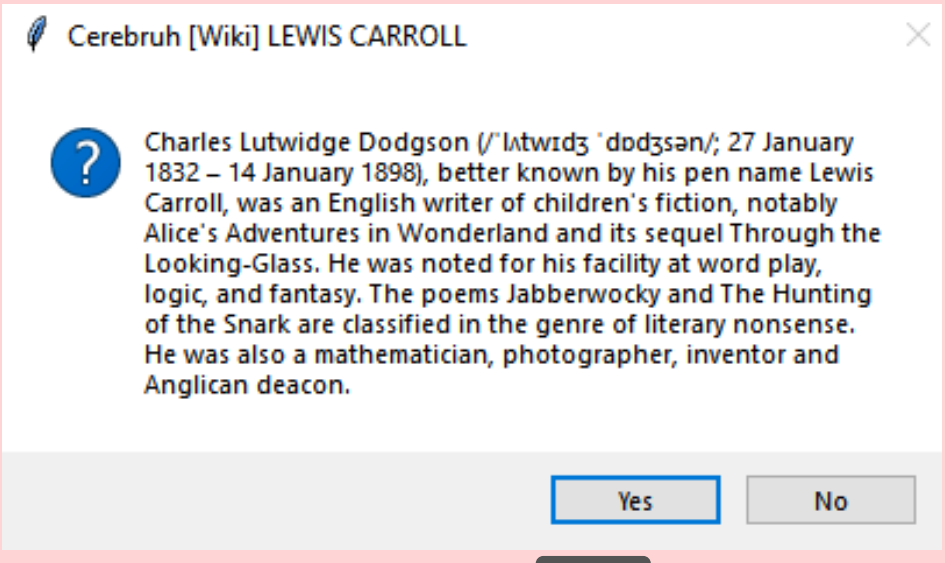
for some laptops,

fn + F2

step 2 Press the F2 key for Wikipedia.

step 3 Sometimes, you might have to press OK to load the paragraph.

step 4 Read.



Note: At STEP 2, F9 can be pressed for the UrbanDictionary version

SOURCES and ACKNOWLEDGEMENTS

Learnt from:

PROGRAMMING FUNDAMENTALS:

Professor Bireswar Das,
IIT Gandhinagar, [ES102].

Jenil Vagadiya, TA,
IIT Gandhinagar, [ES102].

Prajwal Singh, TA,
IIT Gandhinagar, [ES102]

WEBSCRAPING AND REGEX:

geeksforgeeks.com
stackexchange.com
docs.python.org

WIKIPEDIA AND URBANDICTIONARY are community driven online information repositories. All information delivered from this program is sourced to their respective sources. Links are provided in the terminal whenever invoked.