ASASIN 파일 구성

만약에 새로운 컴퓨터에서 실행을 할 경우

- Python 3.5버전을 설치합니다.
- Cmd창에서 다음의 라이브러리들을 추가적으로 설치합니다.
- beautifulsoup, selenium, pymssql, gspread, oauth2client, PyOpenSSL
- 설치방법은 cmd창에서 다음과 같은 명령을 하면 됩니다.
  pip install beautifulsoup4
  pip install selenium
  pip install pymssql
  pip install gspread
  pip install oauth2client
  pip install PyOpenSSL
- 구글 스프레드 시트에 접속할 때 필요한 파일 이름은
  HICASIN-ff20424f976a.json
  입니다. 해당 json파일은 ASASIN.py와 같은 위치에 두어야 합니다.
- Selenium을 통해서 구글 크롬 브라우저를 사용하기 때문에 크롬드라이버를 깔아야 합니다. 크롬드라이버는…
  " C:₩chromedriver.exe" 가 되도록 설치해주시면 됩니다.

```
from bs4 import BeautifulSoup as bs
from selenium import webdriver
import time
import re
import pymssql


Collection_URL = list()
Collection_URL_Error = list()

# MS-SQL에 저장되어있는 검사가 완료된 URL들을 가져온다.
conn = pymssql.connect(server='112.216.50.83', user='kimjh', password='kimjh1234!', database='test_new')
cur = conn.cursor()
```

Import 구문은 파이썬에 필요한 라이브러리를 호출할 때 사용합니다.

bs4는 '뷰티플숩'이라는 HTML을 탐색할 수 있도록 정리해주는 라이브러리 입니다.
selenium은 웹브라우저를 제어할 수 있게해주는 라이브러리 입니다.
re는 정규표현식을 사용하기 위한 라이브러리 입니다.
pymssql은 파이썬을 통해서 MS-SQL을 제어할 수 있게 해주는 라이브러리 입니다.

conn = pymssql.connect(server=?, user=?, password=?, database=?) 를 통해서 SQL서버에 접속합니다.
그 뒤 cur = conn.cursor()를 만들어줍니다. cur변수는 나중에 쿼리 구문을 작동할 때 사용됩니다.

우선, test_new 데이터베이스의 amz_url 테이블에서 AVAIL_URL에 해당하는 주소만을 가져오도록 합니다.

```
cur.execute("SELECT MAX(URL_SEQ) from AMZ_CD") # ASASIN
time.sleep(1)
max_urlseq = cur.fetchone()[0]
if max_urlseq == None:
    max_urlseq = 0

cur.execute("SELECT MAX(seq) from amz_url") # GSUC
time.sleep(1)
max_seq = cur.fetchone()[0]
if max_seq == None:
    max_seq = 0

if max_urlseq >= max_seq:
    conn.close()
    print("새로 업데이트 된 URL이 없습니다.")
    raise SystemExit
else:
    number_of_updated = max_seq - max_urlseq
    print("약 ",number_of_updated,"개 정도의 새로운 URL이 발견되었습니다.")

cur.execute("SELECT seq, url_name, PG FROM amz_url WHERE error_check='AVAIL_URL' AND seq >"+str(max_urlseq))
Collection_URL = cur.fetchall()
print(len(Collection_URL),"개의 주소를 가져왔습니다.")
conn.close()
```

cur.execute("쿼리입력")은 이전에 로그인한 SQL서버에 쿼리를 보내는 기능을 합니다. 이 때 입력하려는 쿼리는 쌍따옴표("")안에 넣어서 입력해야합니다.

cur.execute()를 실행이 성공적으로 끝나면 쿼리에 해당되는 데이터를 불러오게 됩니다. 이 때 **cur.fetchone()**이라는 구문을 실행하여야 이 데이터를 사용할 수 있습니다. 이 때 불러온 데이터는 list()형태로 반환됩니다. fetchone()의 경우에는 데이터 중 가장 먼저의 하나의 데이터만을 반환하고, fetchall()을 사용할 경우 모든 데이터를 반환합니다.

우선, 구글 스프레드 시트에 보면 수집한 URL주소의 순서번호가 있습니다. 이것을 앞으로 URL번호라고 하겠습니다. ASASIN.py에서 ASIN코드를 수집하면서 어떤 URL번호에서 수집을 했는지 같이 저장을 합니다. 따라서 GSUC.py에서 구글 스프레드 시트의 URL을 수집하면서 저장한 URL번호의 최대치와 ASASIN.py에서 수집한 URL번호의 최대치를 비교합니다.

따라서 만일 둘의 최대치가 같거나 ASASIN.py의 URL번호의 최대치가 더 클 경우 코드의 실행을 멈춥니다. GSUC의 URL번호 최대치가 더 큰 경우에는 해당되는 URL번호의 URL주소를 불러와서 스크래퍼를 실행하게 됩니다.

가져온 주소들은 Collection_URL이라는 리스트 형태로 저장이 됩니다.

```
driver = webdriver.Chrome("C:₩chromedriver.exe")
print("브라우저 구동 완료")
```

2페이지에서 설명을 했던 selenium 라이브러리를 통해서 구글 크롬 브라우저를 실행하는 구문 입니다.

구글 크롬 드라이버를 통해서 실행이 되는데, 이 프로그램은 구글에서 다운로드 받을 수 있습니다.
https://sites.google.com/a/chromium.org/chromedriver/downloads

```
# 다음 페이지로 넘어가는 URL을 실행한다.
def Next_Page_Move():
    Next_Page_URL = list_page.find("a",{"title":"Next Page"})["href"]
    driver.get("https://www.amazon.com" + Next_Page_URL)

# HTML 정보를 가져온다.
def Get_HTML():
    global HTML_
    HTML_ = driver.page_source

# 현재페이지의 URL을 저장한다.
def Original_URL():
    global Home_URL
    Home_URL = driver.current_url
```

global 은 함수 내에서 사용된 변수가 함수 밖에서도 유지 될 수 있도록 해주는 선언이다. 이 선언을 하지 않을 경우에는 함수 내에서 만들어진 변수는 함수가 종료된 후 사라지게 된다.

HTML 정보를 가져온다.
: 크롬 브라우저를 통해서 수집한 URL주소로 처음 접속을 한 뒤 뜨는 아마존닷컴 페이지의 HTML을 가져옵니다. 해당 HTML내용을 HTML_변수에 저장합니다.

현재페이지의 URL을 저장한다.
: 설명과 같이, 현재 페이지의 URL을 저장합니다. 한 페이지에 있는 상품의 ASIN을 모두 수집하고 다음페이지로 넘어가기 이전에, 상품들이 전시 되어있는 페이지로 다시 돌아가기 위하여 그 페이지의 주소를 저장하기 위해 사용합니다.

다음 페이지로 넘어가는 URL을 실행한다.
: 각 상품 목록 페이지 하단을 보면 목록 2,3,4...이 있습니다. List_page는 이 각각의 숫자들에 해당하는 URL주소가 담긴 HTML을 따로 때어놓은 변수 입니다. Find구문을 사용할 경우 조건에 해당하는 가장 먼저의 내용을 가져오게 됩니다. 따라서 2가 가장 먼저이므로 2의 URL주소를 가져오고 driver.get을 통해서 해당 URL로 이동합니다.

```
# 상품목록 페이지의 HTML을 분석한다.
def Get_list_from_page(trigger):
    global item_page
    global list_page
    list_page = bs(trigger,"html.parser")
    try:
        item_page = list_page.findAll("li",{"class":"s-result-item  celwidget "})
        if len(item_page)==0:
            raise NameError
    except:
        item_page = list_page.find("ul", {"id":"s-results-list-atf"}).contents
```

아마존 웹사이트는 HTML 구성이 여러 종류로 되어있다. 탐색을 해본 결과 90% 정도가 태그로 이루어져 있거나 태그로 이루어져 있었다. 그 외에도 다른 구성이 있지만 수가 적어서 스크래퍼가 예외처리를 하도록 하였다.

상품목록 페이지의 HTML을 분석한다.

이전에 Get_HTML()을 통해서 상품이 전시되어 있는 페이지의 HTML을 가져왔다. 이제는 Get_list_from_page()를 통해서, Get_HTML을 사용하여 가져온 HTML을 파싱하는 작업을 진행하게 된다.

beautifulsoup라이브러리를 통해서 HTML을 파싱 하게 된다. 앞서 말했듯이 HTML구조가 <li> 또는 로 이루어져 있기 때문에 잘못된 태그를 조건으로 접근을 할 경우에는 에러가 발생하거나 전혀 수집되지 않는다. 이 경우에는 전혀 수집되지 않기 때문에 수집된 것이 없는 경우(len(item_page)==0)에는 NameError를 일으켜서 except구문이 실행되도록 하였습니다.

Get_list_from_page()를 통해서는 전시된 각 상품의 HTML을 파싱한다.

```
# 페이지에 있는 상품들의 URL을 수집한다.
def Get_URL_of_items(trigger):
    global URL_bucket
    URL_bucket = list()
    range_of_list = len(item_page)
    for n in range(range_of_list):
        item_URL = item_page[n].find("a",{"class":"a-link-normal a-text-normal"})['href']
        if len(re.findall("/gp/slredirect/.",item_URL)) > 0 :
            item_URL = "https://www.amazon.com/" + item_URL
        URL_bucket.append(item_URL)

# 상품으로 들어간다.
def Go_into_the_item(trigger):
    driver.get(trigger)
```

페이지에 있는 상품들의 URL을 수집한다.

Get_list_from_page()를 통해서 각각의 상품에 대한 HTML을 파싱하였다. Get_URL_of_items로부터는 이 파싱된 HTML에서 상품페이지로 들어가는 URL주소를 찾아서 저장한다.

상품페이지로 들어가는 URL주소도 두 가지로 혼재한다. 하나는 온전히 www.amazon.com으로 시작하는 URL이고 다른 하나는 /gp/slredirect/로 시작하는 URL이다. 따라서 이는 정규표현식을 사용하여 URL이 /gp/slredirect/로 시작하는지 여부를 가린 다음에 상품페이지로 이동하는 URL을 저장하였다.

검사된 URL은 URL_bucket이라는 리스트 구조를 가진 변수에 저장되도록 하였다.

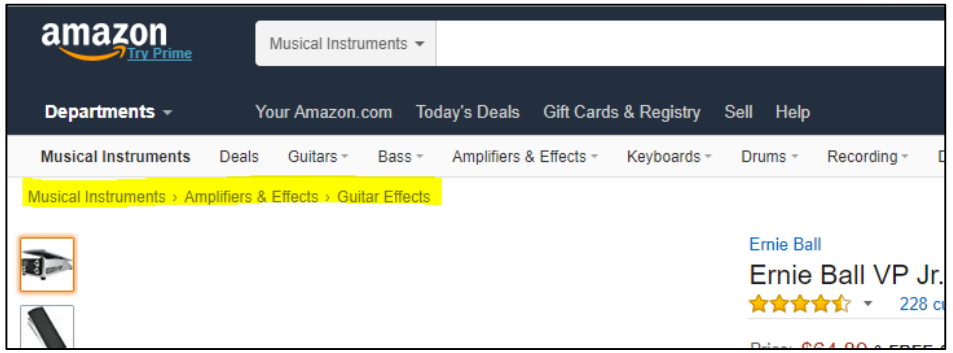
상품으로 들어간다.

URL_bucket에 저장되어있는 상품URL로 접속을 하기 위해서 만든 함수이다.

```
# HTML을 가져온 뒤 필요한 정보를 파싱한다.

def Category_scraper_sub3(trigger): #이걸로 쓰는 것을 권장한다. - 현재 쓰고 있는 함수
    global category
    global node_category
    product_cat = trigger.find("ul",{"class":"a-unordered-list a-horizontal a-size-small"})
    node_name = list()
    node_number = list()
    node_category = list()
    try:
        category_order = product_cat.findAll("li")
        category_node  = product_cat.findAll("a",{"class":"a-link-normal a-color-tertiary"})
        for n_ in category_node:
            node_ = re.findall('node=[0-9]+',n_['href'])[0]
            node_ = node_.split('=')[1]
            node_number.append(node_)
        for c_ in category_order:
            chunk = c_.find("span").get_text().strip().replace("'","`")
            if len(chunk) > 1:
                node_name.append(chunk)
            else:
                continue
        category_zip = list(zip(node_name, node_number))
        for e_ in category_zip:
            node_category.append(e_)
        category = "/".join([x[0] for x in node_category])
        print(category)

    except:
        Category_Error_URL = driver.current_url
        item_category_fail_URL.append(Category_Error_URL)
        print("오류")
        category = 'No Category'
```



위의 그림에서 노란 형광펜이 칠해진 곳에 있는 카테고리 문자열을 가져오게 된다. 각각의 카테고리명은 고유의 노드번호를 가지고 있다.

우선 정규표현식('node=[0-9]+')을 사용하여 노드 번호를 가져온다. 그리고 이 노드번호는 node_number라는 리스트 구조의 변수에 저장을 한다.

다음에는 beautifulsoup을 사용하여 카테고리명을 가져온다. 다만, 나중에 SQL에 넣는 작업을 할 때 따옴표(')가 쿼리 구문에서 문제를 일으키기 때문에 이를 replace()함수를 사용해서 어퍼스트로피(`)로 바꿔주었다. 이렇게 정리된 카테고리명은 node_name이라는 리스트구조의 변수에 저장을 한다.

마지막으로는 node_number와 node_name 각각의 동일한 인덱스 선상에 있는 데이터를 각각 하나의 셋으로 묶어주기 위하여 zip함수를 사용하였다.
예시 : [(bicycle, 999), (car, 919), ...]

이런 형식으로 묶인 카테고리 리스트는 node_category에 저장된다.

가끔씩 상품 페이지에 카테고리가 기재 되어 있지 않은 경우가 있어서 어쩔 수 없이 try/except 구문을 추가하게 되었다.

```python
# ASIN코드 뽑아오는 함수 01
def Parsing_function(): # li일 경우 사용
    print("수집 중...")
    product_asin = product_page.findAll("li")
    Category_scraper_sub3(product_page)
    if category == "No Category":
        print("오류")
    else:
        temp_asin_bucket = list()
        for li in product_asin:
            a = li.get_text()
            b = re.findall('ASIN:+',a)
            try:
                if len(b) > 0:
                    code_asin = re.findall('B[0-1]{1}[0-9]{1}₩w{7}', li.get_text())[0]
                    temp_asin_bucket.append(code_asin)
                    print(code_asin)
            except:
                print("ASIN 에러")
                break
        if len(temp_asin_bucket) != 0:
            code_asin = max(temp_asin_bucket)
            check_and_update(code_asin)
            ASIN_BUCKET.append(code_asin)
```

```python
# ASIN코드 뽑아오는 함수 02
def Parsing_function_sub2(): #tr일 경우 사용
    print("수집 중...")
    product_asin = product_page.findAll("tr")
    Category_scraper_sub3(product_page)
    if category == "No Category":
        print("오류")
    else:
        temp_asin_bucket = list()
        for td in product_asin:
            a = td.get_text()
            b = re.findall('ASIN',a)
            try:
                if len(b) > 0:
                    code_asin = re.findall('B[0-1]{1}[0-9]{1}₩w{7}', td.get_text())[0]
                    temp_asin_bucket.append(code_asin)
                    print(code_asin)
            except:
                print("ASIN 에러")
                break
        if len(temp_asin_bucket) != 0:
            code_asin = max(temp_asin_bucket)
            check_and_update(code_asin)
            ASIN_BUCKET.append(code_asin)
```

상품 페이지도 구성이 와 <ul> 나뉘기 때문에 두 가지 종류의 스크래핑 코드를 작성하였다.

정규표현식을 사용하여 ASIN코드를 가져옵니다. 단, 전체 페이지에 실행을 할 경우 다른 상품의 ASIN코드도 가져오기 때문에 beautifulsoup을 사용하여 Product description란에 있는 ASIN코드를 가져오도록 HTML을 파싱한 뒤 정규표현식을 해당 자료에 사용하였습니다.

```
#카테고리 업데이트 및 ASIN 코드 추가
item_category_essential = ['No Category']
item_storage = [['No Category']]
item_category_fail_URL = [['No Category']]
def check_and_update(trigger):
    if node_category not in item_category_essential:
        item_category_essential.append(node_category)
        item_storage.append([node_category])
    for x in item_category_essential:
        if x == node_category:
            item_storage[item_category_essential.index(node_category)].append(trigger)
```

수집된URL에서 수집한 모든 ASIN과 카테고리는 item_storage에 저장된다.
우선 item_category_essential은 일종의 색인 역할을 한다. 따라서 저장하고자 하는 카테고리가 item_storage에 생성이 되어있는지를 확인할 수 있다. 만약에 item_category_essential에 해당하는 카테고리가 없을 경우 item_storage에도 없으므로, 해당 카테고리를 색인과 item_storage에 추가한다.

쉽게 말하자면, 색인을 찾아봐서 해당 카테고리가 기록이 없다면 데이터 저장소에도 카테고리가 없는 것이므로 색인에 해당 카테고리를 추가하고 데이터 저장소에도 해당 카테고리의 칸을 만드는 것이다.

만일 카테고리가 존재한다면 이전의 함수에서 만들었던 node_categor를 item_storage에 있는 해당 카테고리에 저장을 한다.

```
# 입력한 URL에서 수집한 ASIN코드를 MSSQL DB에 저장하는 코드 - ver.2
def SAVE_ASIN_DB():
    global asin_example
    import datetime
    import pymssql
    conn = pymssql.connect(server = '112.216.50.83', user = 'kimjh', password = 'kimjh1234!', database = 'test_new')
    cursor = conn.cursor()
    for category_ in item_storage:
        date_ = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        seq_ = str(Start_URL[0])
        for asin_ in category_[1:]:
            cursor.execute("SELECT COUNT(GODS_CD) FROM AMZ_CD WHERE GODS_CD='"+asin_+"'")
            duplicate_check = int(cursor.fetchone()[0])
            if duplicate_check > 0:
                print("중복이라 넘어감")
                continue
            else:
                print(asin_, date_, seq_, category_[0][0][0],category_[0][0][1],category_[0][-1][0],category_[0][-1][1])
                print("INSERT INTO AMZ_CD values('"+asin_+"','"+date_+"','"+seq_+"','"+category_[0][0][0]+"','"+category_[0][0][1]+"','"+category_[0][-1][0]+"','"+category_[0
                cursor.execute("INSERT INTO AMZ_CD values('"+asin_+"','"+date_+"','"+seq_+"','"+category_[0][0][0]+"','"+category_[0][0][1]+"','"+category_[0][-1][0]+"','"+ca
    conn.commit()
    conn.close()
    note_for_gspread_success(seq_)
```

수집된URL에서 모은 카테고리와 ASIN 정보를 SQL서버에 저장한다.
카테고리를 저장할 때에는 가장 첫 번째와 가장 마지막 번째의 카테고리명과 노드번호가 저장됩니다.
예를 들어 sports > bicycle > accessories > handles 인 경우에는 sports와 handles만 저장이 됩니다.

마지막으로 SQL에 execute한 내용을 저장하기 위해선 **conn.commit()** 구문을 반드시 실행하여야 합니다.
그 다음, conn.close()로 DB와의 연결을 종료해주는 것이 좋습니다.

맨 마지막에 있는 note_for_gspread_success()는 ASIN을 성공적으로 수집한 뒤 모두 저장을 했을 경우에
구글스프레드시트의 해당 URL 행에 "ASIN수집완료"라는 문구를 표시해줍니다.

```
flag = 0
print(Collection_URL)
for Start_URL in Collection_URL[:]:
    print(Start_URL,type(Start_URL))
    global ASIN_BUCKET
    ASIN_BUCKET = list()
    driver.get(Start_URL[1])
    Total_Target_Page = int(Start_URL[2])


    print("amz_url 테이블의 seq : ",Start_URL[0],"...인 주소 접속 완료.")
    for count_ in range(Total_Target_Page):
        Original_URL()
        Get_HTML()
        time.sleep(1.5)
        Get_list_from_page(HTML_)
        time.sleep(1.5)
        try:
            Get_URL_of_items(item_page)
        except:
            print("상품 목록을 가져오는 도중 오류가 발생했습니다.")
            print("다음 URL로 넘어갑니다.")
            break
        time.sleep(1.5)
        print(count_+1," 번째 페이지에 있는 상품을 수집합니다.......")
        counter = 0
        len_of_ASIN_BUCKET = len(ASIN_BUCKET)
        for x in URL_bucket:
            try:
                Go_into_the_item(x)
            except:
                Collection_URL_Error.append(x)
            HTML_ITEM = driver.page_source
            product_page = bs(HTML_ITEM,"html.parser")

            if product_page.find("div",{"id":"prodDetails"}) == None:
                Parsing_function()
            else : Parsing_function_sub2()



        print(count_+1," 번째 페이지 완료.")
        driver.get(Home_URL)
        try :
            Next_Page_Move()
        except:
            print("페이지 못 넘어감")
            continue
        print("다음 페이지로 넘어갑니다. ------> Page.",count_+2)
    flag += 1
    SAVE_ASIN_DB()

    item_category_essential = ['No Category'] # 리스트 내용 초기화
    item_storage = [['No Category']] # 리스트 내용 초기화
    item_category_fail_URL = [['No Category']] # 리스트 내용 초기화
    print("지정한 주소 및 페이지에서의 수집활동을 모두 수행하였습니다.", "₩n[ 총 ",len(ASIN_BUCKET),"개의 ASIN 번호 수집. ]")


if len(Collection_URL_Error)>0:
    print(Collection_URL_ERROR)
conn.close()
driver.close()
print("브라우저 종료")
```

지금까지 설명한 사용자정의함수를 하나로 엮은 코드 입니다.

순서는 다음과 같습니다.

1. Collection_URL에 담긴 수집한URL을 접속한다.

2. 접속한 페이지(상품들이 전시되어있는)의 각 상품별 URL을 수집한다.

3. 수집한 상품별 URL을 접속한다.

4. 상품페이지에서 카테고리와 ASIN을 가져온 뒤 우선 임시적으로 리스트에 저장한다.

5. 수집한URL에 있는 모든 상품페이지를 돌고 나면 리스트에 있는 수집한 데이터를 MS-SQL서버에 저장한다.

6. 다음 수집한URL로 접속한다.

(--- 반복 ---)

7. 모든 수집한URL을 돌고 난 뒤에는 MS-SQL 그리고 브라우저와의 접속을 끊는다.

GSUR 파일 구성

GSUR 설명 이전에....

- 구글 스프레드 시트에 접속하여 새롭게 올라온 URL이 있는지의 여부를 검사하고, 새로운 URL이 있다면 해당 URL이 ASASIN에서 돌아갈 수 있는지의 여부를 확인한 다음 MSSQL에 저장하는 파일 입니다.

- 검사하는 코드는 ASASIN의 코드와 동일하므로 설명은 생략하였습니다.
  따라서 GSUR에만 있는 코드 위주로 설명을 하였습니다.

- 각 URL마다 검사하는 상품의 개수는 다음과 같습니다.
  식: (한 페이지의 총 상품수)//7 + 3

```
import gspread
from oauth2client.service_account import ServiceAccountCredentials
print("구글 스프레드 시트에 연결하는 중 입니다.")
scope = ['https://spreadsheets.google.com/feeds']
credentials = ServiceAccountCredentials.from_json_keyfile_name('C:₩₩Users₩₩HIC₩₩Desktop₩₩ASIN₩₩HICASIN-ff20424f976a.json',scope)
gc = gspread.authorize(credentials)
wks = gc.open("상품수동수집")
wks = wks.get_worksheet(2)
print("구글 스프레드 시트 연결 완료")
print("DB에 저장된 데이터 수 세는 중....")
conn = pymssql.connect(server = '112.216.50.83', user = 'kimjh', password = 'kimjh1234!', database = 'test_new')
cur = conn.cursor()
cur.execute("SELECT count(seq) from amz_url")
time.sleep(1)
data_count = cur.fetchone()[0]
if data_count == None or data_count == 0:
    data_count = 0
conn.close()
print("데이터 수 확인 완료. DB를 다시 닫습니다.")
```

import gspread ~ print("구글 스프레드 시트 연결 완료")까지의 코드는 파이썬을 통해서 구글스프레드시트에 접속하는 것 입니다. 구글스프레드 시트 파일 중 "상품수동수집"에 접속합니다.

그 이후의 코드는 MSSQL의 test_new.db의 amz_url테이블에 있는 수집된URL의 최대치를 구하는 것 입니다.
(위의 그림에는 "SELECT count(seq) from amz_url"로 되어있지만 "SELECT MAX(seq) from amz_url"로 수정하였습니다.)

만약에 최대치가 없는 경우에는 0으로 설정되도록 하였습니다.

```
NOCAT_URL = list()
ERROR_URL = list()
AVAIL_URL = list()

print("구글 스프레드 시트에 있는 URL을 가져오는 중 입니다.")
for nrow_ in range(data_count+7,wks.row_count):
    row_ = wks.row_values(nrow_)
    if len(row_[0]) == 0:
        break
    elif len(row_[6]) > 0 or len(row_[5]) > 0:
        continue
    elif "https://www.amazon.com/s/ref=sr" in row_[8]:
        wks.update_cell(nrow_,7,"수집 완료")
        Collection_URL.append([row_[0],row_[8]])
    else:
        wks.update_cell(nrow_,7,"ERROR_URL")
        ERROR_URL.append([row_[0],row_[8]])
print("수집 완료")
print("URL 검증을 시작합니다.")
```

구글 스프레드시트에 자료가 입력되어 있는 행과 열의 숫자를 고려하여 URL최대치에 맞추어 새로운 URL을 수집할 수 있도록 하였습니다.

만약에 구글스프레드 시트에 '무게 주의 해주세요'와 같은 주의사항을 적는 칸에 글자가 쓰여 있거나 '수집완료'가 쓰여 있는 칸에 글자가 쓰여 있다면 해당 URL은 수집을 하지 않도록 하였습니다.

그리고 URL의 순서번호가 적혀 있지 않을 때에는 수집활동을 멈추도록 하였습니다.

마지막 else 구문의 경우에는, 이전에 말했던 아마존URL의 90%이상을 차지하는 <li>, <ul> 태그로 구성된 HTML의 경우에는 주소가 https://www.amazon.com/s/ref=sr로 시작하였습니다. 따라서 이 양식으로 시작하지 않는 URL의 경우에는 ASASIN으로 스크래핑을 할 수 없기 때문에 ERROR_URL로 표시하도록 하였습니다.

```
# 사용가능한 URL을 MS-SQL DB에 저장하는 코드
def SAVE_ASIN_DB():
    import datetime
    conn = pymssql.connect(server='112.216.50.83', user='kimjh', password='kimjh1234!', database='test_new')
    cur = conn.cursor()
    date_ = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    for info_ in AVAIL_URL:
        cur.execute("INSERT INTO amz_url (seq, url_name, error_check, date_time, PG) values('"+info_[0]+"','"+info_[1]+"','AVAIL_URL','"+date_+"','"+pgn+"')")
        conn.commit()
    for info_ in ERROR_URL:
        cur.execute("INSERT INTO amz_url (seq, url_name, error_check, date_time, PG) values('"+info_[0]+"','"+info_[1]+"','ERROR_URL','"+date_+"','"+pgn+"')")
        conn.commit()
    for info_ in NOCAT_URL:
        cur.execute("INSERT INTO amz_url (seq, url_name, error_check, date_time, PG) values('"+info_[0]+"','"+info_[1]+"','NOCAT_URL','"+date_+"','"+pgn+"')")
        conn.commit()
    conn.close()
```

MS-SQL서버에 접속하여 수집한 데이터를 저장하는 코드입니다.

ASASIN에서 사용 가능한 URL은 AVAIL_URL로서,
ERROR를 일으켰거나 그럴 소지가 있는 URL은 ERROR_URL로서,
카테고리가 수집되지 않는 URL은 NOCAT_URL로서 SQL서버에 저장되도록 하였습니다.

```
    if (len(item_storage[0])-1) > (len(URL_bucket)//7+3)*0.8 and flag == 0:
        print("No Category 오류")
        NOCAT_URL.append(Start_URL) ############################## NOCAT_URL ##########
    elif len(Collection_URL_Error) > (len(URL_bucket)//7+3)*0.8 or flag == 1:
        print('상품URL오류')
        ERROR_URL.append(Start_URL) ############################## ERROR_URL ##########
    else :
        print("수집 가능한 URL")
        AVAIL_URL.append(Start_URL) ############################## AVAIL_URL ##########

    item_category_essential = ['No Category'] # 리스트 내용 초기화
    item_storage = [['No Category']] # 리스트 내용 초기화
    item_category_fail_URL = [['No Category']] # 리스트 내용 초기화
    print("수집한 주소에 대한 검사를 모두 수행하였습니다.")
    print(len(AVAIL_URL)+len(ERROR_URL)+len(NOCAT_URL),"개의 주소를 검사.")
    print("AVAIL_URL ",len(AVAIL_URL),"개")
    print("ERROR_URL ",len(ERROR_URL),"개")
    print("NOCAT_URL ",len(NOCAT_URL),"개")
print("수집한 URL을 DB에 저장합니다.")
SAVE_ASIN_DB() ############################################## MS-SQL DB에 저장 #######
print("저장 완료")
driver.close()
print("브라우저 종료")
```

If문 : 만일 수집된URL의 리스트에서 'No category에 해당하는 상품의 URL'이 전체 상품 개수의 80%이상을 차지할 경우 그리고 flag==0에 해당할 경우 NOCAT_URL로 분류하였습니다.

Elif문 : 만일 수집된URL의 리스트에서 Error를 일으킨 URL에 해당하는 상품의 URL이 전체 상품 개수의 80% 이상을 차지하거나 flag==1에 해당할 경우 ERROR_URL로 분류하였습니다.

Else문 : 만일 이 모든 조건에 걸리지 않는 경우에는 AVAIL_URL로 분류하였습니다.

웹스크래퍼 사용방법

1

- IDLE (Python 3.5 64-bit)로 GSUR.py를 연다.
- F5를 누른다.

GSUR.py로 구글 스프레드시트에 새로운 URL이 올라왔나 탐색을 하고 만일 있다면 해당 URL이 ASASIN에서 돌아갈 수 있는지 검사를 한 뒤 사용가능여부와 함께 MSSQL서버(테이블명 : amz_url)에 저장합니다.

2

- IDLE (Python 3.5 64-bit)로 ASASIN.py를 연다.
- F5를 누른다.