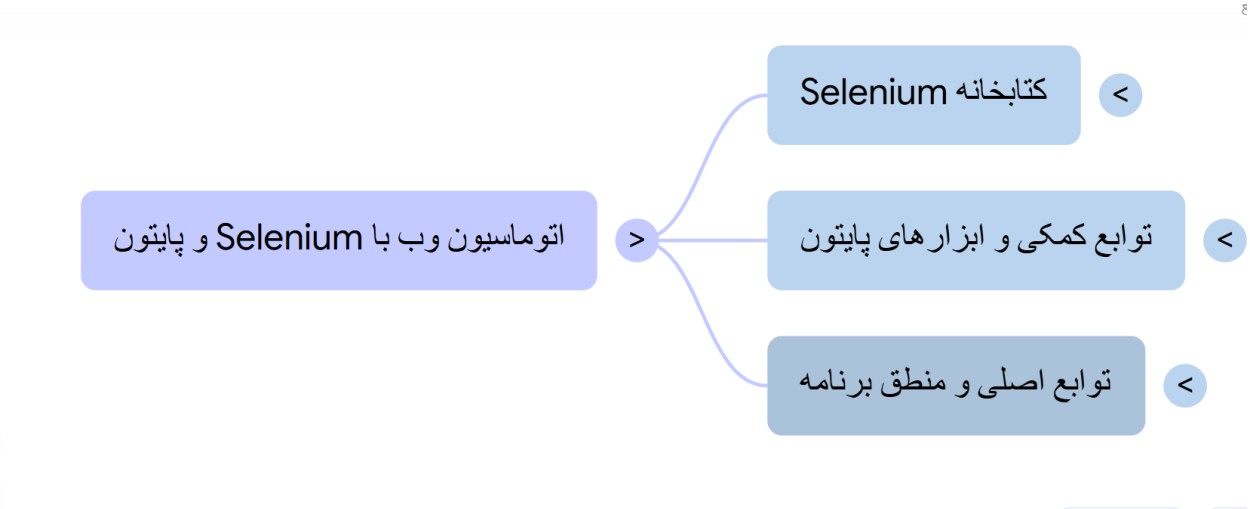
این سند مجموعه‌ای از **کدهای پایتون** را توضیح می‌دهند که از کتابخانه **Selenium** برای **اتوماسیون وب** و **استخراج داده‌ها** استفاده می‌کنند. هدف اصلی این کدها **تست خودکار برنامه‌های کاربردی وب** و **وب اسکرپینگ** است. توابع مختلفی برای **مدیریت مرورگر**)مانند راه‌اندازی و بستن(، **تعامل با عناصر صفحه**)از جمله اسکرول و کلیک کردن بر روی دسته‌بندی‌ها و زیردسته‌ها(، **استخراج اطلاعات** از صفحات وب (مانند نام و مقادیر ویژگی‌ها)، و **ذخیره‌سازی پیشرفت و نتایج** در فایل‌های Excel و JSON تعریف شده‌اند. همچنین، این کدها شامل **مدیریت خطاها و تلاش‌های مجدد** برای اطمینان از پایداری عملیات اتوماسیون هستند.

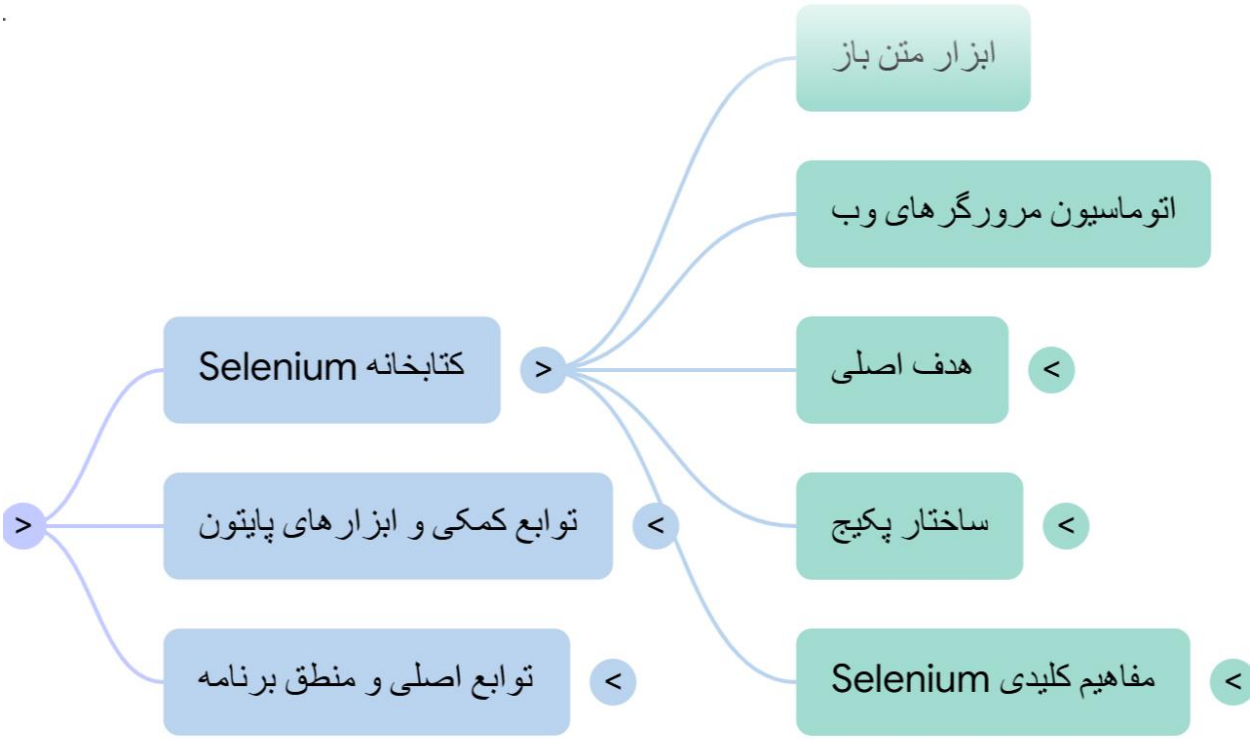


کتابخانه selenium

Selenium یک ابزار متن باز(کتابخانه) است که برای اتوماسیون مرورگر های وب استفاده میشود .
هدف اصلی آن تست خودکار برنامه های کاربردی وب (web application testing) است مثلا :

- وب اسکرپینگ(web scraping) : استخراج خودکار داده ها از وب سایت
- تست خودکار
- اتوماسیون وظایف تکراری در تعامل با وب .

کل کتابخانه selenium به صورت یک پکیج اصلی (package selenium) ارائه میشود که شامل چندین زیرپکیج (subpackage) و ماژول های مستقل است




```
from selenium import webdriver
```

این خط کد پکیج اصلی Selenium Webdriver را وارد میکند (import) و به ما امکان کنترل مرورگرهای وب زیر پکیج ها (مثل firefox و chrome و edge) را میدهد .

```
Selenium / کتابخانه

----------Webdriver / پکیج اصلی (کنترل مرورگرها)
```

----------Chrome /

----------Firfox /

----------Edge /

----------Common /

----------rimote /

----------support / و.....

```
from selenium.webdriver.chrome.service import Service
```

selenium /

---------webdriver /

----------chrome / زیر پکیج مخصوص کروم

----------webdriver.py ماژول

----------options.py ماژول

----------service.py ماژول

```
from selenium.webdriver.common.by import By
```

```
selenium /
---------webdriver/
------------------common /
------------------------by.py  ماژول
----------------------------
```

```
from selenium.webdriver.support.ui import WebDriverWait
```

```
selenium /
---------webdriver /
------------------support /
------------------------- ui.py  ماژول
```

```
from selenium.webdriver.support import expected_conditions as EC
```

```
selenium /
---------webdriver /
------------------support /
------------------------- expected_conditions.py  =   EC
```

مجموعه ای از پیش شرط های آماده برای تعامل با صفحات وب

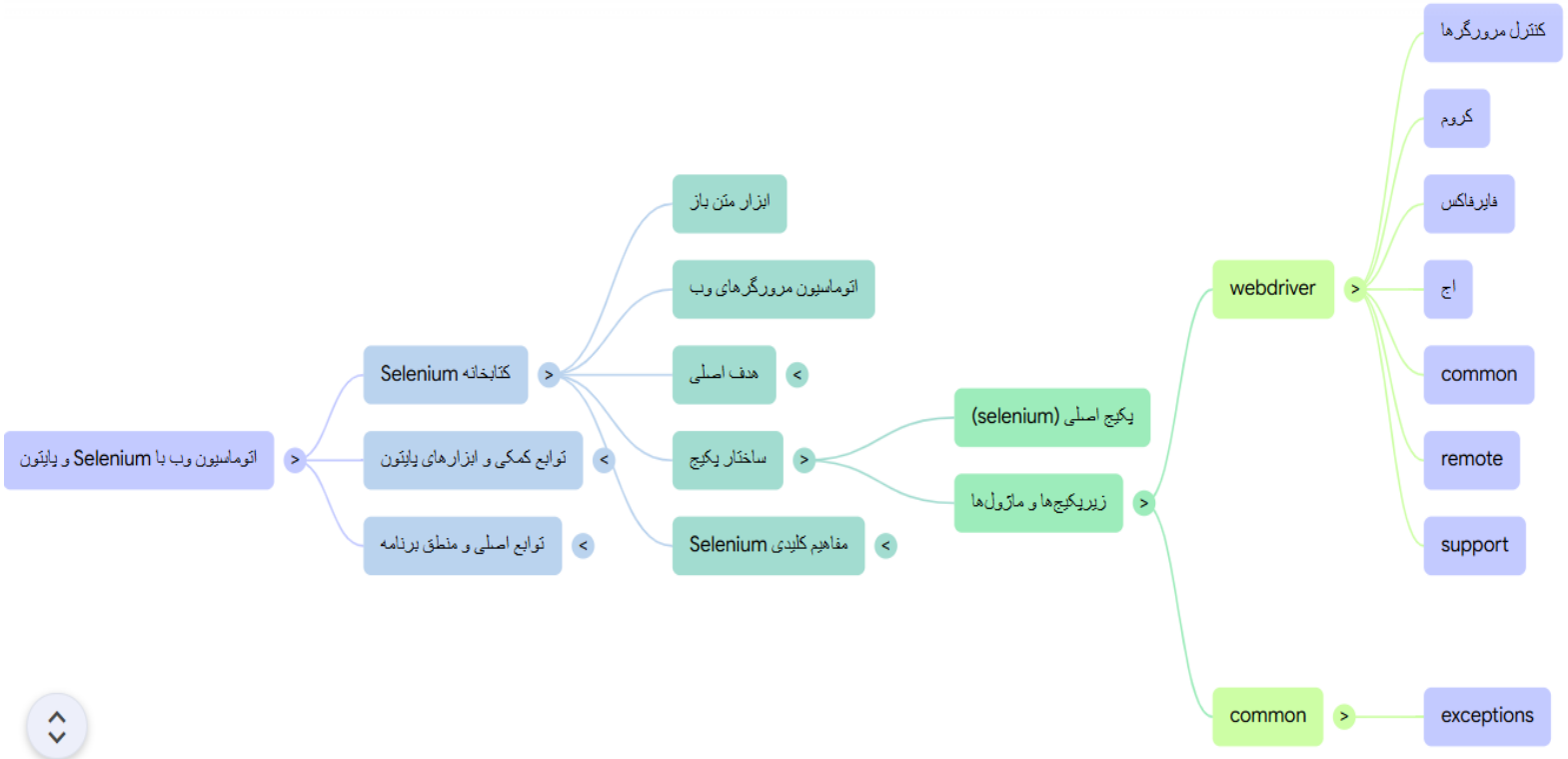
```
from selenium.common.exceptions import TimeoutException,
StaleElementReferenceException, ElementClickInterceptedException
```

```
selenium /
--------- common /
----------------exception.py  ماژول
----------------------  TimeoutException    کلاس
```

---------- StaleElementReferenceException

---------- ElementClickInterceptedException

برای خطایابی و مدیریت خطا استفاده میشوند .




```
ctypes.windll.kernel32.SetThreadExecutionState(0x80000002)
```

از کتابخانه SetThreadExecutionState در پایتون ، تابع ctypes در ویندوز فراخوانی میکند .

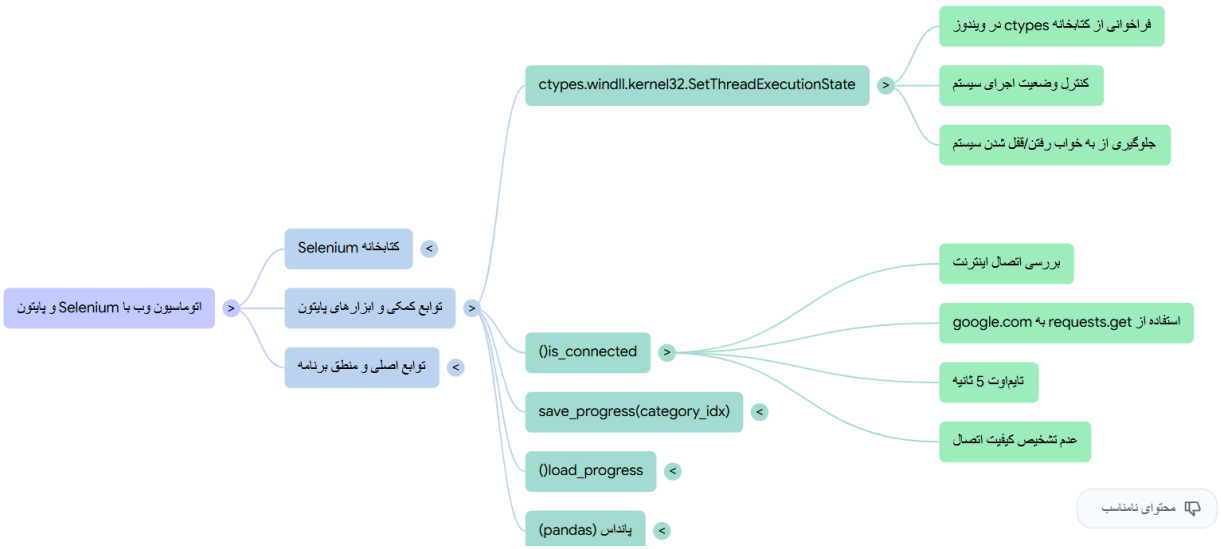
این تابع برای کنترل وضعیت اجرای سیستم (با kernel32.dll مانع به خواب رفتن سیستم یا قفل شدن سیستم میشود) استفاده میشود .

توضیح پارامتر (0x80000002) به سیستم ویندوز میگوید نمایشگر رو فعال نگه دار و این وضعیت را تا زمانیکه تغییری ایجاد نشده حفظ کن .

```python
def is_connected():
    try:
        requests.get('https://www.google.com', timeout=5)
        return True
    except requests.ConnectionError:
        return False
```

بررسی اتصال اینترنت :

برای تست اتصال google انتخاب شده است ; اگر سرور در 5 ثانیه پاسخی دریافت نکند ، درخواست لغو میشود . عدم تشخیص کیفیت اتصال (پایداری ، سرعت اتصال)




```python
def save_progress(category_idx):
    with open('progress.json', 'w') as f:
        json.dump({'category_idx': category_idx}, f)
```

ذخیره پیشرفت برنامه :

فایل ایجاد شده یک فایل json هست (progress.json)

Write ---- >’w’ اگر فایل وجود داشته باشد ، محتوای آن پاک میشود واز اول نوشته می شود .

مثال : { “category_idx ”: 3 } دیکشنری

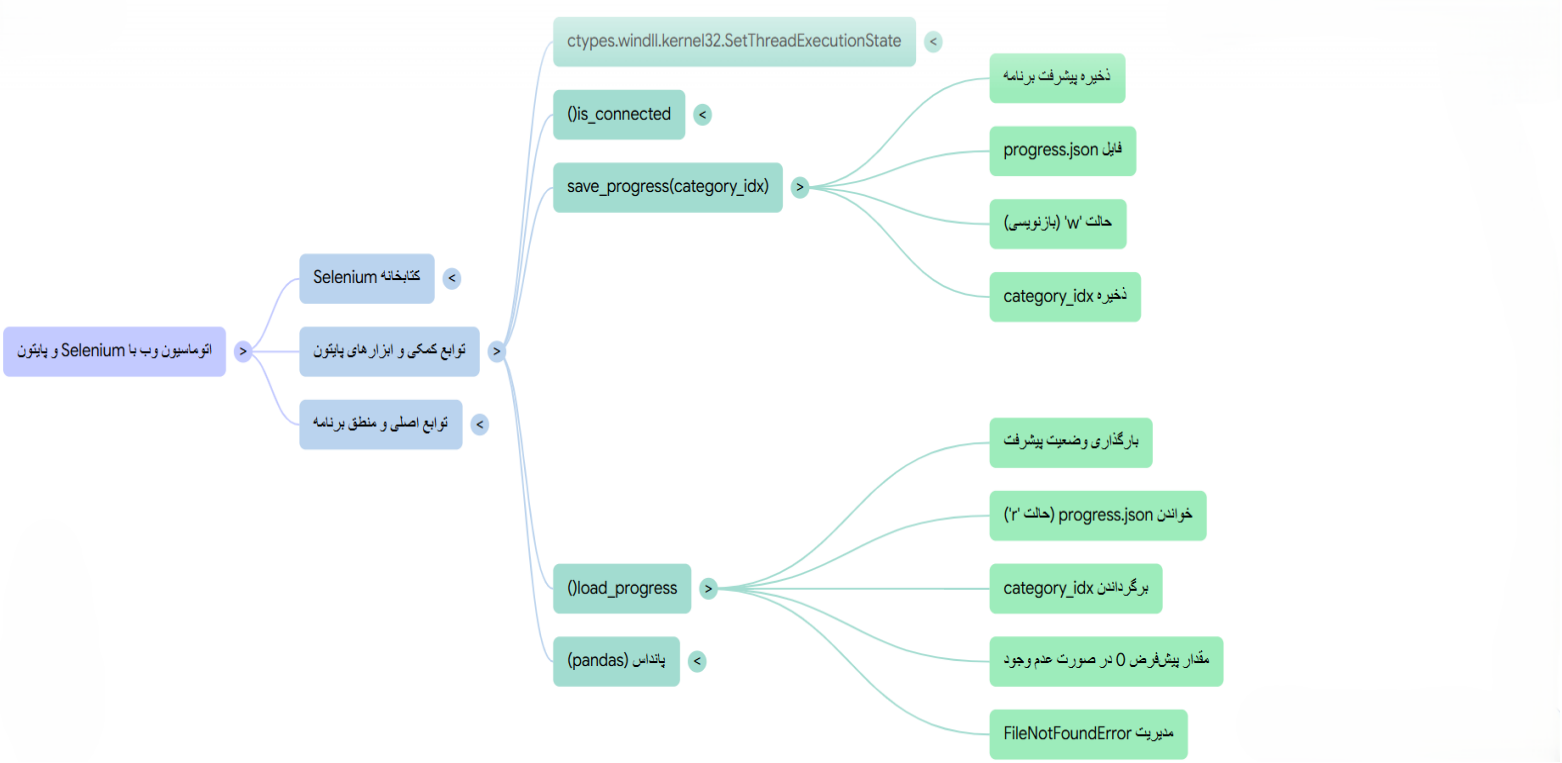
```
def load_progress():
    try:
        with open('progress.json', 'r') as f:
            progress = json.load(f)
            return progress.get('category_idx', 0 )
    except FileNotFoundError:
        return 0
```

تابع بارگذاری وضعیت پیشرفت برنامه :

فایل Progress.json درحالت Read باز میشه و محتوای فایل json خوانده میشود

این خط سعی می‌کنه مقدار مربوط به کلید **‘Catrgory_IDX’** رو از دیکشنری Progress بخواند .اگر کلید در دیکشنری موجود باشه مقدار را برمیگرداند ولی اگر مقداری نداشت 0 فرض میشود

FileNotFoundError خطای خاص expert هست که درصورت پیدا نشدن فایل json در مسیر که برنامه اجرا میشود .



```python
# راه‌اندازی مرورگر
def setup_driver(chromedriver_path):

    service = Service(executable_path=chromedriver_path)

    driver = webdriver.Chrome(service=service)

    return driver
```

تابع راه اندازی مرورگر :

Chromedriver_path یک پارامتر ورودی برای تابع است .

Chromedriver مترجم بین selenium و مرورگر chrome است . وقتی میخواهیم با استفاده از selenium مرورگر chrome را کنترل کنیم (مثلا صفحه ای باز کنیم یا فعالیتی انجام دهیم) selenium به chromedriver دستور میدهد و chromedriver این دستورات را به مرورگر chrome ارسال میکند و نتیجه را در نهایت به seleniumبرمیگرداند .

چرا اصلا به chromedriver نیاز داریم ؟ selenium به تنهایی قادر نیست که مرورگر chrome کنترل کند برای هر مرورگری که بخواهیم با selenium کنترلی انجام دهیم به یک "درایور" مخصوص همان مرورگر نیاز داریم مثلا :

برای Firefox به geckodriver و برای مرورگر Edge هم به msedgedriver نیاز داریم .

نکته مهم : نسخه chromedriver باید با نسخه مرورگر chrome که روی سیستم نصب شده سازگار باشد .

```python
service = Service(executable_path=chromedriver_path)
```

service اول یه متغیر پایتون هست .
Service دوم یک کلاس از کتابخانه selenium است کد زیر برای import کردن این کلاس هست :

```python
from selenium.webdriver.chrome.service import Service
```

درواقع توی این متن میگوییم که از داخل پکیج selenium ، بعد از زیر پکیج webdriver و بعد از ان زیر پکیج chrome.service ، کلاس service برای من import کن .

Executable_path یک نام استاندارد در selenium برای مشخص کردن مسیر فایل اجرایی درایور است .

Chromedriver_path این متغیر مقداری که در خود نگه داشته است همان مسیر فایل اجرایی chromedriver است .

```
driver = webdriver.Chrome(service=service)
```

driver =

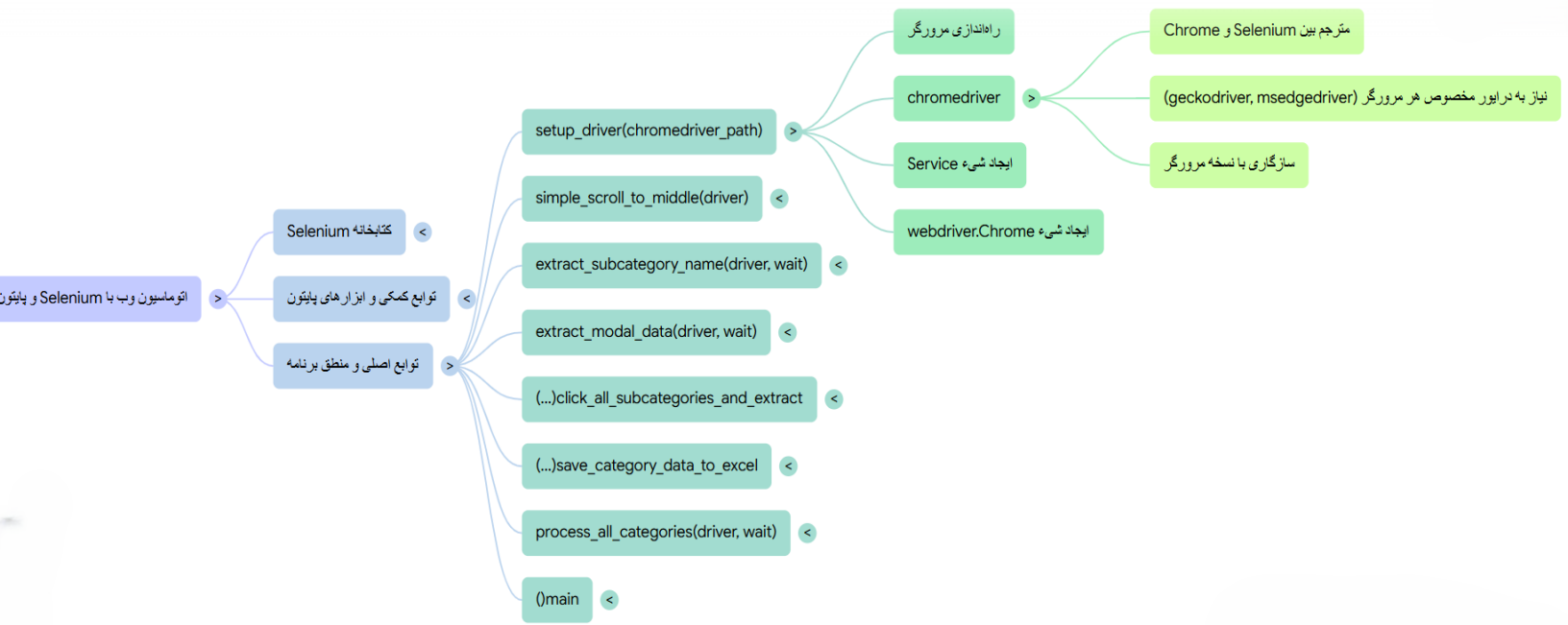
ما یک شیء جدید از کلاس webdriver.Chromeمی‌سازیم که همون مرورگر واقعی ماست.

این شیء به ما امکان می‌ده که مرورگر رو کنترل کنیم (باز کردن صفحات، کلیک کردن، نوشتن متن، و...).

webdriver.Chrome(...)

webdriverهمون پکیج اصلی Selenium هست.

Chromeکلاس مخصوص مرورگر Google Chrome هست.



service=service

اینجا ما شیء service که در خط قبلی ساختیم رو به مرورگر پاس می‌دیم.

یعنی به webdriver.Chrome می‌گیم: "برای اجرای مرورگر، از این درایوری که آماده کردم استفاده کن."

```
def simple_scroll_to_middle(driver):
    total_height = driver.execute_script("return
document.body.scrollHeight")
    target_pos = total_height / 2
    driver.execute_script(f"window.scrollTo(0, {target_pos});")
    time.sleep(1.5)
```

(driver)

این پارامتر ورودی تابع هست .همون شیء مرورگر از نوع webdriver.Chrome هست که در واقع مرورگر رو کنترل می‌کنه.

در اینجا به تابع گفته می‌شه که از driver برای انجام اسکرول صفحه استفاده کنه.

`total_height = driver.execute_script("return document.body.scrollHeight")`

total_height =

ما یک متغیر به نام total_height تعریف می‌کنیم که قرار هست ارتفاع کل صفحه وب رو نگه داره.

driver.execute_script(...)

این متد از webdriver هست که به ما این امکان رو می‌ده که یک اسکریپت JavaScript رو داخل مرورگر اجرا کنیم.

به عبارتی، این تابع برای تعامل با وب‌سایت از طریق جاوااسکریپت استفاده می‌کنه.

"return document.body.scrollHeight"

این بخش یک اسکریپت JavaScript هست که به مرورگر می‌گه: "ارتفاع کل صفحه (از بالا تا پایین) رو به من بده."

document.body.scrollHeight به ارتفاع کل صفحه در حالت اسکرول شده اشاره داره.

---

target_pos = total_height / 2

target_pos =

اینجا متغیر جدیدی به نام target_pos تعریف می‌کنیم که موقعیت هدف برای اسکرول رو ذخیره می‌کنه.

total_height / 2

ما ارتفاع کل صفحه رو تقسیم بر ۲ می‌کنیم تا به وسط صفحه برسیم. به این ترتیب، target_pos مقدار ارتفاع نقطه وسط صفحه رو ذخیره می‌کنه.

---

driver.execute_script(f"window.scrollTo(0, {target_pos});")

driver.execute_script(...)

این دوباره همون متد برای اجرای JavaScript داخل مرورگره.

f"window.scrollTo(0, {target_pos});" ◆

اینجا یک اسکریپت JavaScript دیگه نوشته شده (window.scrollTo(0, {target_pos} .به مرورگر می‌گه که صفحه رو از نقطه‌ی (0, target_pos)اسکرول کنه.

0 به معنای حرکت به سمت چپ صفحه است (چون اسکرول عمودی هست).

{target_pos }هم همون نقطه وسط صفحه هست که قبلاً محاسبه کردیم.

این دستور باعث می‌شه که صفحه به نقطه وسط اسکرول بشه.

---

time.sleep(1.5)

این خط از کتابخانه time استفاده می‌کنه و باعث می‌شه که برنامه برای 1.5 ثانیه توقف کنه.

این کار معمولاً برای اطمینان از این که اسکرول به درستی انجام شده و صفحه کمی زمان برای لود کردن داده‌ها داشته باشه، مناسب هست .

```python
# استخراج نام زیر دسته
def extract_subcategory_name(driver, wait):
    attempts = 0
    max_attempts = 3
    while attempts < max_attempts:
        try:
            wrapper =
wait.until(EC.presence_of_element_located((By.CSS_SELECTOR,
"div.callery_calculate_wraper")))
            h4 = wrapper.find_element(By.TAG_NAME, "h4")
            name = h4.text.strip()
            if name:
                print(f"نام زیر دسته استخراج شد: {name}")
                return name
            else:
                print("نام زیر دسته خالی است، تلاش مجدد...")
                attempts += 1
                time.sleep(0.5)
        except Exception as e:
            print(f"خطا در استخراج نام زیر دسته: {e}")
            attempts += 1
            time.sleep(0.5)
    print("نام زیر دسته پس از چند تلاش خالی ماند، مقدار 'نامشخص' قرار داده شد.")
    return "نامشخص"
```

1. تعریف تابع :

def extract_subcategory_name(driver, wait):

extract_subcategory_name

اسم تابع هست که به معنی "استخراج نام زیر دسته" هست.

(driver, wait)

driver : این پارامتر یک شیء از نوع webdriver هست که مرورگر رو کنترل می‌کنه.

wait : این پارامتر یک شیء از نوع WebDriverWait هست که به ما این امکان رو می‌ده که برای حضور یک عنصر در صفحه صبر کنیم.

---

2. تعداد تلاش‌ها و محدودیت :

attempts = 0

max_attempts = 3

attempts = 0

تعداد تلاش‌های فعلی برای استخراج نام زیر دسته رو شروع می‌کنیم از 0.

max_attempts = 3

تعداد حداکثر تلاش‌ها رو برای استخراج نام زیر دسته تعیین می‌کنیم. در اینجا ۳ تلاش مجاز داریم.

---

3. حلقه برای تلاش مجدد:

while attempts < max_attempts:

while attempts < max_attempts:

این حلقه اجرا میشه تا زمانی که تعداد تلاش‌ها از max_attempts کمتر باشه.

این به این معنیه که اگر نام زیر دسته به درستی استخراج نشه، تلاش‌های بعدی انجام می‌شه.

---

4. تلاش برای استخراج نام زیر دسته:

try:

این بخش از کد برای مدیریت استثناها (Exceptions) استفاده میشه. یعنی اگر کدی که داخل try هست به اشتباه بخوره یا خطا بده، به بخش except می‌ره.

---

4. پیدا کردن عنصر مورد نظر:

```
wrapper = wait.until(EC.presence_of_element_located((By.CSS_SELECTOR, "div.callery_calculate_wraper")))
```

wrapper =

متغیر wrapper برای ذخیره‌ی عنصر HTML مورد نظر استفاده می‌شود.

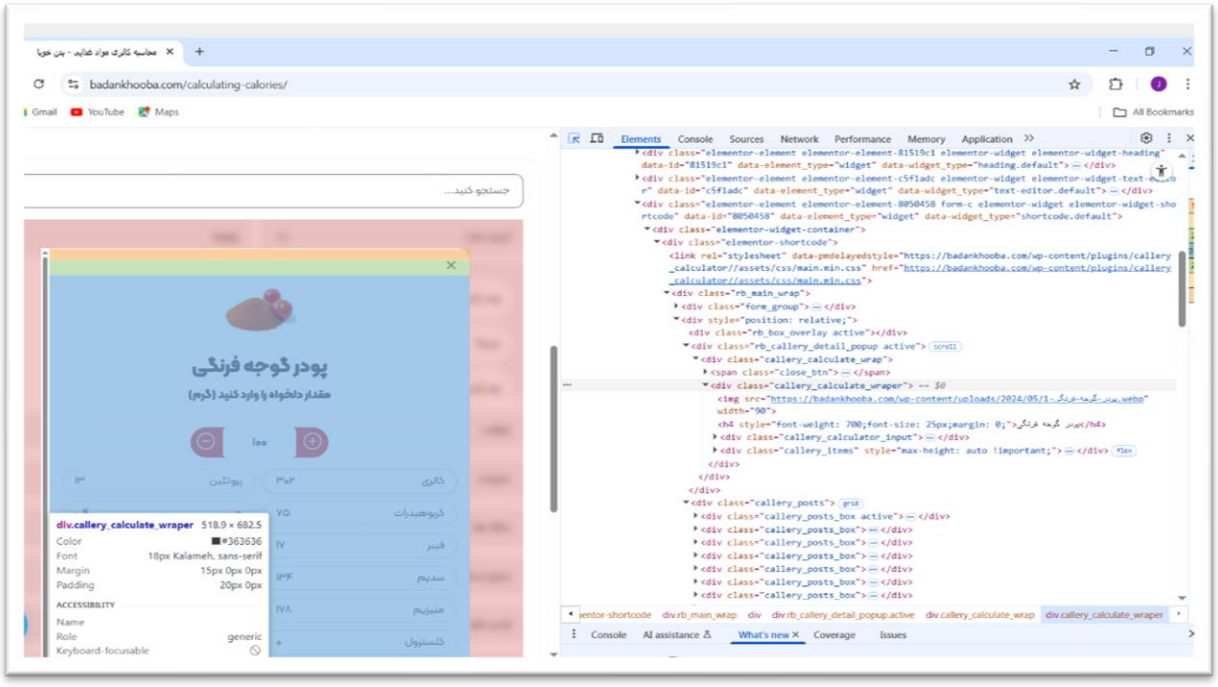
wait.until(...)

از WebDriverWait برای صبر کردن به مدت معین تا عنصر مدنظر در صفحه پیدا بشه استفاده می‌کنیم.

EC.presence_of_element_located به معنی "وجود عنصر در صفحه" هست.

(By.CSS_SELECTOR, "div.callery_calculate_wraper")

این کد به دنبال یک عنصر div با کلاس callery_calculate_wraper می‌گرده. این عنصر باید در صفحه وب موجود باشه تا بتوانیم ادامه بدیم.



6.استخراج نام زیر دسته:

```
h4 = wrapper.find_element(By.TAG_NAME, "h4")

name = h4.text.strip()
```

```
h4 = wrapper.find_element(By.TAG_NAME, "h4")
```

در این خط، داخل wrapper که قبلاً پیدا کردیم، به دنبال تگ h4 می‌گردیم. این تگ معمولاً برای نمایش عنوان یا نام‌ها استفاده می‌شود.

```
name = h4.text.strip()
```

h4.text متن داخل تگ h4 رو به دست میاره.

strip() برای حذف فاصله‌های اضافی در ابتدا و انتهای متن استفاده میشه.

7.چک کردن نام و تصمیم‌گیری:

```
if name:
```

اگر name مقدار غیر خالی داشته باشه، یعنی نام زیر دسته استخراج شده. در این صورت:

print(f"نام زیر دسته استخراج شد: {name}")

نام زیر دسته را در کنسول چاپ می‌کنیم.

return name

نام استخراج شده را به عنوان خروجی تابع باز می‌گردونیم.

else:

اگر name خالی باشه (مثلاً تگ h4 هیچ متنی نداشته باشه)

print("نام زیر دسته خالی است، تلاش مجدد...")

پیام خطا چاپ می‌کنیم که نام خالی است و دوباره تلاش می‌کنیم.

attempts += 1

تعداد تلاش‌ها را یکی افزایش می‌دهیم.

time.sleep(0.5)

0.5 ثانیه صبر می‌کنیم تا دوباره درخواست انجام بشه. این تأخیر ممکنه برای جلوگیری از درخواست‌های مکرر یا جهت اطمینان از لود کامل صفحه باشد.

8.مدیریت استثناها:

```
except Exception as e:
    print(f "خطا در استخراج نام زیر دسته: {e} ")
    attempts += 1
    time.sleep(0.5)
```

except Exception as e:

اگر در زمان استخراج نام زیر دسته خطا رخ بده، وارد بخش except می‌شویم.

print(f "خطا در استخراج نام زیر دسته: {e} ")

پیام خطای مربوطه چاپ می‌شود که می‌تواند شامل اطلاعات دقیق از خطا باشد.

attempts += 1

تعداد تلاش‌ها رو یکی افزایش می‌دهیم.

time.sleep(0.5)

بعد از خطا، 0.5 ثانیه صبر می‌کنیم تا دوباره تلاش کنیم.

9.پایان تلاش‌ها و مقدار پیش‌فرض:

print("نام زیر دسته پس از چند تلاش خالی ماند، مقدار 'نامشخص' قرار داده شد.")

return "نامشخص"

print("نام زیر دسته پس از چند تلاش خالی ماند، مقدار 'نامشخص' قرار داده شد.")

اگر بعد از ۳ تلاش (یا تعداد مشخص شده) نام استخراج نشد، پیامی چاپ می‌کنیم که نام هنوز خالی مانده است.

return "نامشخص"

در نهایت اگر نام استخراج نشد، مقدار پیش‌فرض 'نامشخص' را باز می‌گردونیم.

```
# استخراج داده‌های مودال
def extract_modal_data(driver, wait):
    results = []
    try:
        subcat_name = extract_subcategory_name(driver, wait)

        callery_items =
wait.until(EC.presence_of_all_elements_located((By.CLASS_NAME,
"callery_item")))

        for item in callery_items:
            try:
```

```
                item_name = item.find_element(By.XPATH,
".//span[not(@class)]").text
            except:
                item_name = "نامشخص"
            try:
                item_value_str = item.find_element(By.CLASS_NAME,
"callery_item_value").text
                item_value_float = float(item_value_str)
            except:
                item_value_float = 0.0

            print(f"ویژگی : {item_name}  , مقدار : {item_value_float}")
            results.append((item_name, item_value_float))
            time.sleep(0.3)

        close_btn =
wait.until(EC.element_to_be_clickable((By.CLASS_NAME, "close_btn")))
        close_btn.click()
        print("مودال بسته شد")
        time.sleep(1)

    except Exception as e:
        print(f"خطا در استخراج یا بستن مودال: {e}")

    return subcat_name, results
```

extract_modal_data

اسم تابع که به معنی "استخراج داده‌های مودال" هست. تابع برای استخراج اطلاعات از یک مودال (پنجره پاپ‌آپ) در صفحه وب طراحی شده.

(driver, wait)

driver: شیء webdriverهست که به ما امکان کنترل مرورگر را می‌ده.

wait: شیء WebDriverWait که برای صبر کردن تا زمانی که یک عنصر در صفحه حاضر بشه استفاده می‌شود.

.2تعریف نتایج:

```
results = []
```

اینجا یک لیست خالی به نام results تعریف می‌کنیم که در آن اطلاعات استخراج شده از مودال ذخیره خواهند شد .

.4استخراج نام زیر دسته:

```
subcat_name = extract_subcategory_name(driver, wait)
```

این خط نام زیر دسته را با استفاده از تابع extract_subcategory_name استخراج می‌کند که قبلاً توضیح داده شده است.

این نام به عنوان اولین داده استخراجی در نظر گرفته می‌شود

.5انتظار برای حضور تمام آیتم‌ها:

```
callery_items=wait.until(EC.presence_of_all_elements_located((By.CLASS_NAME, "callery_item")))
```

callery_items =

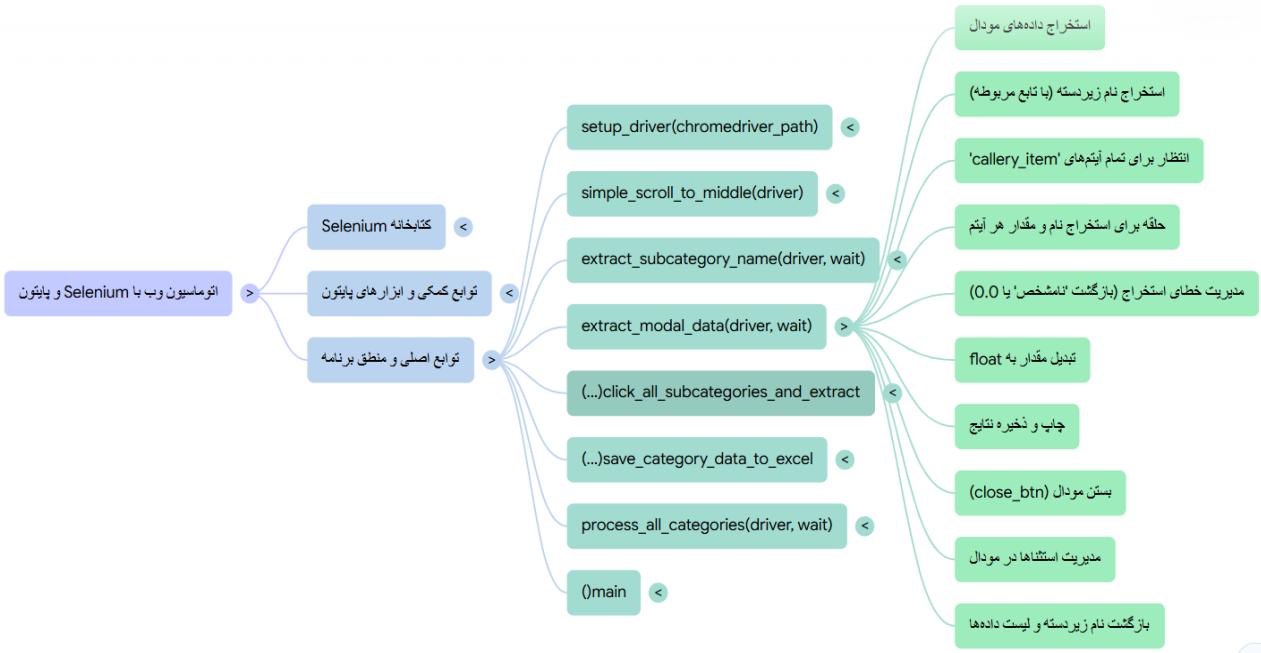
متغیر callery_items برای ذخیره تمام آیتم‌های موجود در مودال استفاده می‌شود.

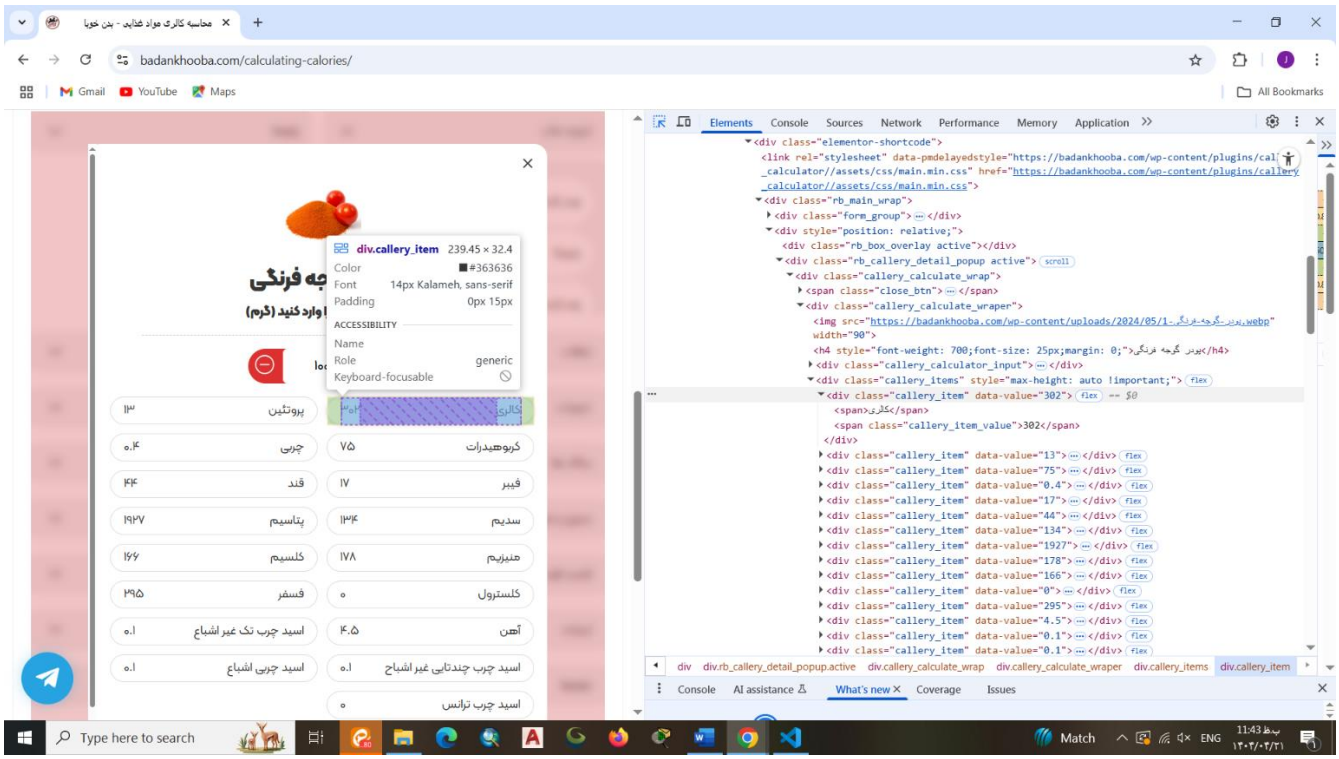
wait.until(...)

با استفاده از WebDriverWaitمنتظر می‌مانیم تا همه آیتم‌هایی که دارای کلاس callery_item هستند در صفحه بارگذاری و حاضر بشوند.

(By.CLASS_NAME, "callery_item")

اینجا ما با استفاده از By.CLASS_NAME به دنبال تمام عناصری با کلاس callery_item می‌گردیم.


اتوماسیون وب با Selenium و پایتون
کتابخانه Selenium
توابع کمکی و ابزارهای پایتون
توابع اصلی و منطق برنامه
setup_driver(chromedriver_path)
simple_scroll_to_middle(driver)
extract_subcategory_name(driver, wait)
extract_modal_data(driver, wait)
(...)click_all_subcategories_and_extract
(...)save_category_data_to_excel
process_all_categories(driver, wait)
()main
استخراج داده‌های مودال
استخراج نام زیردسته (با تابع مربوطه)
انتظار برای تمام آیتم‌های 'callery_item'
حلقه برای استخراج نام و مقدار هر آیتم
مدیریت خطای استخراج (بازگشت 'نامشخص' یا 0.0)
تبدیل مقدار به float
چاپ و ذخیره نتایج
بستن مودال (close_btn)
مدیریت استثناها در مودال
بازگشت نام زیردسته و لیست داده‌ها


badankhooba.com/calculating-calories/
div.callery_item 239.45 × 32.4
Color #363636
Font 14px Kalameh, sans-serif
Padding 0px 15px
ACCESSIBILITY
Name
Role generic
Keyboard-focusable
Elements Console Sources Network Performance Memory Application
Type here to search

.6استخراج نام و مقدار هر آیتم:

```
for item in callery_items:
    try:
        item_name = item.find_element(By.XPATH, ".//span[not(@class)]").text
    except:
        item_name = "نامشخص"
```

for item in callery_items:

این حلقه برای پیمایش تمام آیتم‌هایی است که داخل مودال پیدا کردیم.

item_name = item.find_element(By.XPATH, ".//span[not(@class)]").text

برای هر آیتم، تلاش می‌کنیم که نام آن را استخراج کنیم. به دنبال یک عنصر span بدون کلاس (not(@class))می‌گردیم.

text.برای استخراج متن داخل آن عنصر استفاده می‌شود.

except:

اگر در زمان استخراج نام آیتم خطا رخ بده، وارد بخش except می‌شویم.

item_name = "نامشخص"

در صورتی که نام آیتم پیدا نشد یا خطایی رخ داد، مقدار "نامشخص "به عنوان نام آیتم قرار می‌دهیم.

.7استخراج مقدار هر آیتم:

```
    try:
        item_value_str = item.find_element(By.CLASS_NAME,
"callery_item_value").text
        item_value_float = float(item_value_str)
    except:
        item_value_float = 0.0
```

item_value_str = item.find_element(By.CLASS_NAME, "callery_item_value").text

این خط برای استخراج مقدار آیتم است. به دنبال عناصری با کلاس callery_item_value می‌گردیم.

text.متن داخل عنصر را می‌گیرد.

item_value_float = float(item_value_str)

مقدار استخراجی به رشته (string) تبدیل شده و سپس به نوع float تبدیل می‌شود تا برای پردازش‌های عددی مناسب باشد.

except:

اگر در زمان استخراج مقدار آیتم خطا رخ دهد (مثلاً عنصر مورد نظر پیدا نشود)، وارد بخش except می‌شویم.

item_value_float = 0.0

در صورتی که خطا رخ دهد، مقدار پیش‌فرض 0.0 به عنوان مقدار آیتم در نظر گرفته می‌شود.

---

8. ذخیره نتایج:

```
print( f "ویژگی : {item_name} , مقدار : {item_value_float} ")
results.append((item_name, item_value_float))
time.sleep(0.3)
```

print(f"ویژگی : {item_name} , مقدار : {item_value_float}")

نام و مقدار هر آیتم را در کنسول چاپ می‌کنیم.

◆ results.append((item_name, item_value_float))

نام و مقدار هر آیتم را به لیست results اضافه می‌کنیم.

◆ time.sleep(0.3)

0.3 ثانیه صبر می‌کنیم تا درخواست‌های مکرر را کنترل کنیم و از ارسال بیش از حد درخواست جلوگیری شود.

.9بستن مودال:

```
close_btn = wait.until(EC.element_to_be_clickable((By.CLASS_NAME, "close_btn")))
close_btn.click()
print("مودال بسته شد")
time.sleep(1)
```

close_btn = wait.until(EC.element_to_be_clickable((By.CLASS_NAME, "close_btn")))

منتظر می‌مانیم تا دکمه بسته شدن مودال با کلاس close_btn قابل کلیک باشد.

close_btn.click()

دکمه بسته شدن مودال را کلیک می‌کنیم.

print("مودال بسته شد")

پیامی چاپ می‌کنیم که نشان می‌دهد مودال بسته شده است.

time.sleep(1)

1ثانیه صبر می‌کنیم تا مطمئن شویم مودال بسته شده و صفحه به حالت عادی بازگشته است.

.10مدیریت استثنا:

```
except Exception as e:
    print(f"خطا در استخراج یا بستن مودال: {e}")
```

except Exception as e:

اگر در هر کدام از مراحل استخراج یا بستن مودال خطا رخ بده، وارد این بخش می‌شویم.

print(f "خطا در استخراج یا بستن مودال : {e}")

پیام خطا چاپ می‌کنیم که اطلاعات دقیق‌تری در مورد نوع خطا می‌دهد.

11.بازگشت داده‌ها:

```
return subcat_name, results
```

نام زیر دسته (subcat_name) و لیست نتایج استخراج شده (results) را به عنوان خروجی تابع برمی‌گردانیم.

```
def click_all_subcategories_and_extract(driver, wait, category_box,
category_idx, category_name):
    all_results = []

    subcategories = category_box.find_elements(By.CLASS_NAME,
"callery_product")
    print(f"تعداد زیر دسته‌ها برای دسته {category_idx + 1}: {len(subcategories)}")

    idx = 0
    max_retry = 5

    while idx < len(subcategories):
        retry = 0
        success = False
        while retry < max_retry and not success:
            try:
                subcategories =
category_box.find_elements(By.CLASS_NAME, "callery_product")
                subcat = subcategories[idx]

                driver.execute_script("""
                    arguments[0].scrollIntoView({behavior: 'smooth',
block: 'center', inline: 'nearest'});
                """, subcat)
                time.sleep(0.5)

                wait.until(EC.element_to_be_clickable(subcat))

                try:
```

```
                    subcat.click()
                    time.sleep(3)
                except Exception:
                    print("کلیک مستقیم موفق نبود، استفاده از جاواسکریپت برای کلیک")
                    driver.execute_script("arguments[0].click();",
subcat)
                    time.sleep(3)

                wait.until(EC.visibility_of_element_located((By.CSS_SE
LECTOR, "div.callery_calculate_wraper")))

                subcat_name, modal_data = extract_modal_data(driver,
wait)

                all_results.append({
                    "category_index": category_idx + 1,
                    "category_name": category_name,
                    "subcategory_index": idx + 1,
                    "subcategory_name": subcat_name,
                    "data": modal_data
                })

                idx += 1
                success = True

            except (StaleElementReferenceException,
ElementClickInterceptedException) as e:
                retry += 1
                print(f"خطا{e} در زیر دسته{idx + 1} ، تلاش مجدد
{retry}/{max_retry}")
                time.sleep(1)
            except Exception as e:
                import traceback
                print(f"خطای غیرمنتظره در زیر دسته{idx + 1}: {repr(e)}")
                traceback.print_exc()
                idx += 1
                success = True

        if not success:
```

```
            print(f"تلاش برای کلیک روی زیر دسته {idx + 1} ناموفق بود، ادامه می‌دهیم.")
            idx += 1

    # بستن دسته بعد از پایان زیر دسته‌ها (کلیک روی عنوان دسته)
    try:
        category_title_element = 
category_box.find_element(By.CLASS_NAME, "callery_posts_box_title")
        driver.execute_script("""
            arguments[0].scrollIntoView({behavior: 'smooth', block: 
'center', inline: 'nearest'});
        """, category_title_element)
        time.sleep(0.3)
        category_title_element.click()
        print(f"دسته {category_idx + 1} (زیر دسته‌ها جمع شدند) بسته شد.")
        time.sleep(0.5)
    except Exception as e:
        print(f"خطا در بستن دسته {category_idx + 1}: {e}")

    return all_results
```

1.تعریف تابع:

```
def click_all_subcategories_and_extract(driver, wait, category_box, category_idx, 
category_name):
```

click_all_subcategories_and_extract

این تابع برای کلیک روی تمام زیر دسته‌ها و استخراج داده‌ها طراحی شده است.

(driver, wait, category_box, category_idx, category_name)

driver : شیء مرورگر برای کنترل مرورگر.

wait : شیء WebDriverWait برای صبر کردن تا زمانی که یک قابل رویت باشد.

category_box : جعبه‌ای که تمام زیر دسته‌ها در آن قرار دارند.

category_idx : شاخص دسته (مربوط به دسته‌ای که در حال حاضر در حال پردازش است).

category_name : نام دسته‌ای که در حال حاضر پردازش می‌شود.

3. استخراج تمام زیر دسته‌ها:

```
subcategories = category_box.find_elements(By.CLASS_NAME, "callery_product")
print(f "تعداد زیر دسته‌ها برای دسته {category_idx + 1} : {len(subcategories)} ")
```

این خط تمام زیر دسته‌ها را با استفاده از find_elements یعنی همه عناصری که با کلاس callery_productتطابق دارند) از جعبه‌ی دسته (category_box) استخراج می‌کند.

تعداد زیر دسته‌ها در کنسول چاپ می‌شود category_idx + 1 .برای نمایش دسته با شماره صحیح (از 1 به جای 0) است.

---

4. حلقه برای پیمایش زیر دسته‌ها:

```
idx = 0
max_retry = 5
```

متغیر idx برای پیگیری موقعیت یا اندیس زیر دسته در حال پردازش استفاده می‌شود.

تعداد تلاش‌های مجاز برای کلیک روی هر زیر دسته در صورت مواجهه با خطا تنظیم می‌شود.

---

5. حلقه برای کلیک و استخراج داده:

```
while idx < len(subcategories):
    retry = 0
    success = False
```

این حلقه برای پیمایش و پردازش تمام زیر دسته‌ها اجرا می‌شود. تا زمانی که idx از طول لیست زیر دسته‌ها کمتر باشد، ادامه پیدا می‌کند.

متغیر retry برای پیگیری تعداد تلاش‌های مجدد در صورت مواجهه با خطا استفاده می‌شود.

.6تلاش برای کلیک روی زیر دسته:

```
while retry < max_retry and not success:
```

```
try:
```

این حلقه داخلی برای تلاش مجدد برای کلیک روی زیر دسته است. اگر کلیک موفق نبود و تعداد تلاش‌ها کمتر از max_retry باشد، دوباره تلاش می‌کند.

.اسکرول به زیر دسته 7:

```
subcategories = category_box.find_elements(By.CLASS_NAME, "callery_product")
```

```
subcat = subcategories[idx]
```

```
driver.execute_script(""" arguments[0].scrollIntoView({behavior: 'smooth', block: 'center', inline: 'nearest ;'})""", subcat)
```

```
time.sleep(0.5)
```

subcat = subcategories[idx]

در این خط، زیر دسته‌ای که می‌خواهیم روی آن کلیک کنیم از لیست subcategories استخراج می‌شود.

driver.execute_script(...)

این خط از JavaScript برای اسکرول به سمت زیر دسته مورد نظر استفاده می‌شود .
scrollIntoView این زیر دسته را به مرکز صفحه می‌آورد.

.

.8انتظار برای کلیک و کلیک روی زیر دسته:

```
wait.until(EC.element_to_be_clickable(subcat))
```

این خط از WebDriverWaitاستفاده می‌کند تا زمانی که عنصر زیر دسته قابل کلیک شود، منتظر می‌ماند.

```
try:
    subcat.click()
    time.sleep(3)
except Exception:
    print("کلیک مستقیم موفق نبود، استفاده از جاوااسکریپت برای کلیک")
    driver.execute_script("arguments[0].click();", subcat)
    time.sleep(3)
```

subcat.click()

تلاش می‌کنیم که روی زیر دسته کلیک کنیم.

except Exception:

اگر کلیک مستقیم موفق نبود، از جاوااسکریپت برای کلیک استفاده می‌کنیم) که معمولا در صورت مشکلات تعامل با DOM استفاده می‌شود.(

.9استخراج داده‌ها از مودال:

```
wait.until(EC.visibility_of_element_located((By.CSS_SELECTOR, "div.callery_calculate_wraper")))
subcat_name, modal_data = extract_modal_data(driver, wait)
```

منتظر می‌مانیم تا مودال (پنجره پاپ‌آپ) با کلاس callery_calculate_wraper قابل مشاهده باشد.

◆ subcat_name, modal_data = extract_modal_data(driver, wait)

نام زیر دسته و داده‌های مودال از طریق تابع extract_modal_data استخراج می‌شود.

10. ذخیره نتایج:

```
all_results.append({
    "category_index": category_idx + 1,
    "category_name": category_name,
    "subcategory_index": idx + 1,
    "subcategory_name": subcat_name,
    "data": modal_data
})
```

نتایج استخراج شده شامل اطلاعات مربوط به دسته، زیر دسته و داده‌های مودال به لیست all_resultsاضافه می‌شود.

---

11. مدیریت خطاها:

```
except (StaleElementReferenceException,ElementClickInterceptedException) as e:
    retry += 1
    print(f"خطا {e} در زیر دسته {idx + 1} ، تلاش مجدد {retry}/{max_retry}")
    time.sleep(1)
```

◆ این خطاها مربوط به مشکلاتی هستند که ممکن است در هنگام تعامل با عنصر پیش بیاید:

StaleElementReferenceException: وقتی عنصر مورد نظر دیگر در دسترس نباشد.

ElementClickInterceptedException: وقتی یک عنصر به دلیل مسئله‌ای در لایه‌های صفحه قابل کلیک نباشد.

---

12. پایان تلاش‌های ناموفق:

if not success:

print(f"تلاش برای کلیک روی زیر دسته {idx + 1} ناموفق بود، ادامه می‌دهیم.")

idx += 1

◆ اگر پس از چند تلاش (max_retry) کلیک موفق نشد، پیامی چاپ می‌شود و به سراغ زیر دسته بعدی می‌رویم.

2.پیدا کردن عنوان دسته:

category_title_element = category_box.find_element(By.CLASS_NAME, "callery_posts_box_title")

category_title_element =

این خط یک متغیر category_title_element برای ذخیره عنصر HTML عنوان دسته تعریف می‌کند.

◆ category_box.find_element(By.CLASS_NAME, "callery_posts_box_title")

در این خط، داخل category_box به دنبال عنصر با کلاس callery_posts_box_title می‌گردیم.

---

3. اسکرول به سمت عنوان دسته:

driver.execute_script("""

arguments[0].scrollIntoView({behavior: 'smooth', block: 'center', inline: 'nearest'});

""", category_title_element)

time.sleep(0.3)

◆ driver.execute_script(...)

از این خط برای اجرای یک اسکریپت JavaScript داخل مرورگر استفاده می‌کنیم.

این اسکریپت باعث می‌شود که عنوان دسته (که در متغیر category_title_element ذخیره کرده‌ایم) به وسط صفحه اسکرول شود.

scrollIntoView به این معناست که عنصر مورد نظر به دید کاربر برسد.

behavior: 'smooth' برای انجام اسکرول به‌صورت نرم و روان است.

block: 'center' به این معناست که عنصر در وسط صفحه قرار گیرد.

.4کلیک روی عنوان دسته:

```
category_title_element.click()
```

این خط روی عنصر عنوان دسته که پیدا کردیم کلیک می‌کند..

.5پیام خروجی:

```
print(f "دسته {category_idx + 1} (زیر دسته‌ها جمع شدند) " بسته شد )
```

پیامی در کنسول چاپ می‌کند که نشان می‌دهد دسته مورد نظر بسته شده است.

category_idx + 1 برای چاپ شماره دسته به صورت صحیح (از 1 به جای 0) استفاده می‌شود.

.7مدیریت استثنا:

```
except Exception as e:
    print(f"خطا در بستن دسته {category_idx + 1}: {e}")
```

اگر خطایی در هر بخش از کدهای بالا رخ دهد، وارد این بخش می‌شویم.

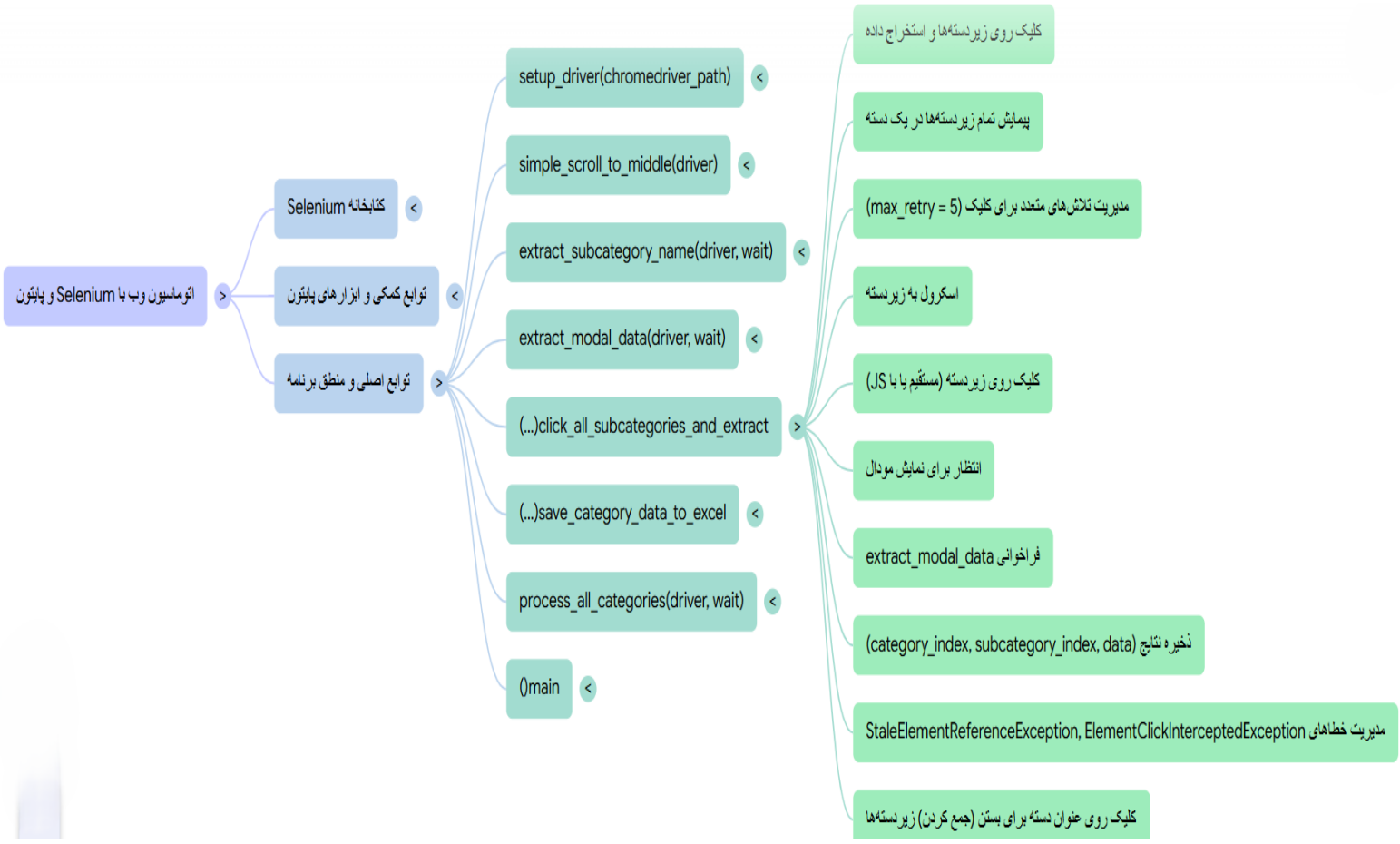
Exceptionبه طور کلی تمام انواع استثناها را می‌گیرد.

پیامی چاپ می‌کند که خطای رخ داده را نمایش می‌دهد. این پیام می‌تواند شامل توضیح خطا باشد که در متغیر eذخیره شده است.

.8بازگشت نتایج:

```
return all_results
```

در نهایت، لیست all_results که احتمالاً شامل نتایج استخراج شده از دسته‌ها و زیر دسته‌ها است، به تابع فراخوانی باز می‌گردد.



```
def save_category_data_to_excel(all_results, category_name,
category_index):
    safe_name = "".join(c for c in category_name if c.isalnum() or c
in (' ', '_')).rstrip()
    filename =
f"calories_data_category_{category_index+1}_{safe_name}.xlsx"
    print(f"ذخیره داده‌های دسته{category_index + 1} با نام '{category_name}' در
فایل{filename}")

    flat_data = []
    for entry in all_results:
        subcat_idx = entry["subcategory_index"]
        subcat_name = entry["subcategory_name"]
        for name, value in entry["data"]:
            flat_data.append({
                "category_index": category_index + 1,
                "category_name": category_name,
                "subcategory_index": subcat_idx,
                "subcategory_name": subcat_name,
                "feature_name": name,
                "feature_value": value
            })

    if not flat_data:
        flat_data.append({
            "category_index": category_index + 1,
```

```
            "category_name": category_name,
            "subcategory_index": None,
            "subcategory_name": None,
            "feature_name": "هیچ داده‌ای یافت نشد",
            "feature_value": None
        })

    df = pd.DataFrame(flat_data)
    df.to_excel(filename, index=False)
    print(f"داده‌ها در فایل {filename} ذخیره شدند.")
```

1.تعریف تابع:

```
def save_category_data_to_excel(all_results, category_name, category_index):
```

این تابع وظیفه ذخیره‌سازی داده‌های یک دسته در فایل Excel را دارد.

(all_results, category_name, category_index)

all_results : داده‌هایی که از زیر دسته‌ها استخراج شده‌اند و به لیست all_results اضافه شده‌اند.

category_name : نام دسته‌ای که داده‌ها برای آن ذخیره می‌شود.

category_index : شاخص دسته در مجموعه داده‌ها (برای ایجاد نام فایل مناسب).

2.ایجاد نام امن برای فایل:

```
safe_name = "".join(c for c in category_name if c.isalnum() or c in (' ', '_')).rstrip()
```

safe_name =

در این خط، یک نام امن برای فایل Excel ایجاد می‌شود.

"".join(c for c in category_name if c.isalnum() or c in (' ', '_'))

این کد هر کاراکتر غیر مجاز را از category_name حذف می‌کند و فقط کاراکترهای الفبایی، عددی، فاصله و _را مجاز می‌شمارد.

به‌عبارت دیگر، هر چیزی که غیر از حروف، اعداد، فاصله و آندرلاین باشد از نام حذف می‌شود.

rstrip()

این متد فضای اضافی در انتهای نام فایل را حذف می‌کند.

.3ایجاد نام فایل:

```
filename = f"calories_data_category_{category_index+1}_{safe_name}.xlsx"
```

filename =

این متغیر برای ذخیره‌سازی نام فایل استفاده می‌شود.

f"calories_data_category_{category_index+1}_{safe_name}.xlsx"

این نام فایل به صورت پویا ساخته می‌شود و شامل:

category_index+1 : شاخص دسته (با شروع از 1) برای ساخت یک نام یکتا.

safe_name : نام امن دسته که در خط قبلی ساخته شد.

.4چاپ پیغام:

```
print(f"ذخیره داده‌های دسته {category_index + 1} با نام '{category_name}' در فایل {filename}")"
```

این خط برای نمایش یک پیام در کنسول است که نشان می‌دهد داده‌ها در حال ذخیره‌سازی در فایل مربوطه هستند .

---

5. مسطح‌سازی داده‌ها:

```
flat_data = []
```

```
  for entry in all_results:
    subcat_idx = entry["subcategory_index"]
    subcat_name = entry["subcategory_name"]
    for name, value in entry["data"]:
      flat_data.append({
        "category_index": category_index + 1,
        "category_name": category_name,
        "subcategory_index": subcat_idx,
        "subcategory_name": subcat_name,
        "feature_name": name,
        "feature_value": value
      })
```

flat_data = []

متغیر flat_data برای ذخیره‌سازی داده‌های مسطح شده (در قالب یک لیست از دیکشنری‌ها) ایجاد می‌شود.

for entry in all_results:

حلقه‌ای برای پیمایش هر ورودی در all_results.

subcat_idx = entry["subcategory_index"]

استخراج شاخص زیر دسته برای هر ورودی.

subcat_name = entry["subcategory_name"]

استخراج نام زیر دسته برای هر ورودی.

for name, value in entry["data"]:

حلقه‌ای برای پیمایش داده‌های داخل هر ورودی (که به صورت جفت نام و مقدار هستند).

flat_data.append(...)

هر جفت داده (ویژگی و مقدار) به لیست flat_data اضافه می‌شود.

این داده‌ها شامل: شاخص دسته، نام دسته، شاخص زیر دسته، نام زیر دسته، نام ویژگی و مقدار ویژگی است.

.6در صورت خالی بودن داده‌ها:

```
if not flat_data:
    flat_data.append({
        "category_index": category_index + 1,
        "category_name": category_name,
        "subcategory_index": None,
        "subcategory_name": None,
        "feature_name": "هیچ داده‌ای یافت نشد",
        "feature_value": None
    })
```

if not flat_data:

اگر داده‌ها در flat_data خالی باشند، یعنی هیچ ویژگی‌ای پیدا نشده است.

flat_data.append(...)

یک دیکشنری با پیغام "هیچ داده‌ای یافت نشد "به flat_dataاضافه می‌شود تا نشان دهد هیچ داده‌ای استخراج نشده است.

.7ساخت DataFrame و ذخیره در Excel:

```
df = pd.DataFrame(flat_data)
df.to_excel(filename, index=False)
print(f"داده‌ها در فایل {filename} ذخیره شدند.")
```

df = pd.DataFrame(flat_data)

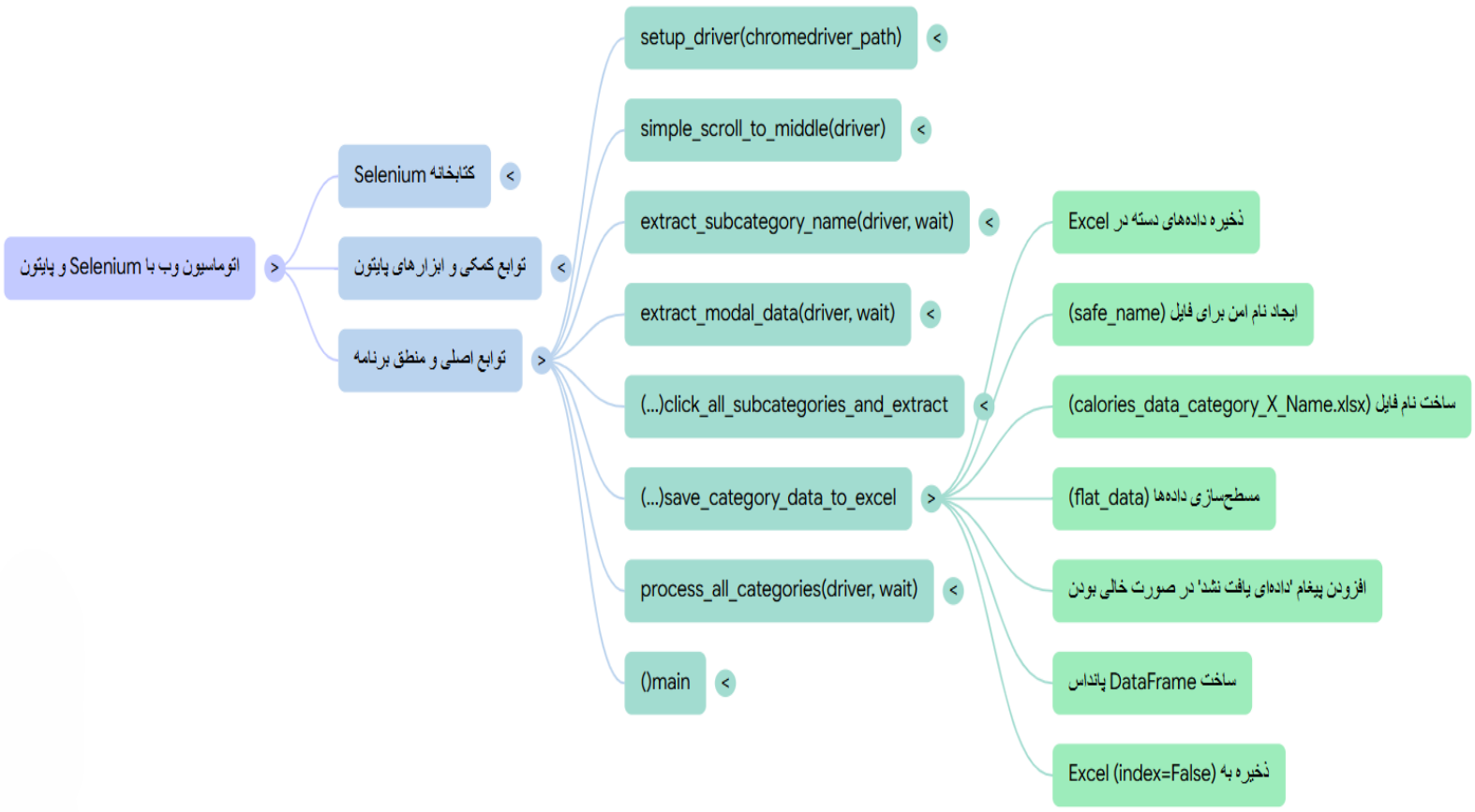
داده‌های مسطح شده به یک DataFrame پانداس تبدیل می‌شود. این ساختار برای ذخیره‌سازی داده‌ها در فایل Excel بسیار مناسب است.

df.to_excel(filename, index=False)

داده‌ها در فایل Excel با نام filenameذخیره می‌شوند.

index=Falseبه این معناست که شاخص DataFrame در فایل Excel ذخیره نشود.

پیامی چاپ می‌شود که نشان می‌دهد داده‌ها با موفقیت در فایل Excel ذخیره شده‌اند.



```
# پردازش تمام دسته‌ها
def process_all_categories(driver, wait):
    wait.until(lambda d: d.execute_script("return
document.readyState") == "complete")
    simple_scroll_to_middle(driver)
    print("اسکرول به وسط صفحه انجام شد.")

    categories =
wait.until(EC.presence_of_all_elements_located((By.CLASS_NAME,
"callery_posts_box")))
    idx = load_progress()  # بارگذاری وضعیت پردازش از فایل

    max_retry_category = 3

    while idx < len(categories):
        retry_category = 0
        success_category = False

        while retry_category < max_retry_category and not
success_category:
            try:
                categories =
wait.until(EC.presence_of_all_elements_located((By.CLASS_NAME,
"callery_posts_box")))
                if idx >= len(categories):
                    print("دسته‌ای باقی نمانده یا به انتها رسیدیم. پایان برنامه.")
                    return

                category_box = categories[idx]
                category_name =
category_box.find_element(By.CLASS_NAME,
"callery_posts_box_title").text.strip()

                driver.execute_script("""
                    arguments[0].scrollIntoView({behavior: 'smooth',
block: 'center', inline: 'nearest'});
                """, category_box)
                time.sleep(0.5)
```

```
                try:
                    category_box.click()
                except ElementClickInterceptedException:
                    driver.execute_script("window.scrollTo(0, 0);")
                    time.sleep(0.5)
                    category_box.click()

                print(f"کلیک روی دسته {idx + 1} انجام شد: {category_name}")

                wait.until(EC.presence_of_element_located((By.CLASS_NA
ME, "callery_product")))

                all_results =
click_all_subcategories_and_extract(driver, wait, category_box, idx,
category_name)

                save_category_data_to_excel(all_results,
category_name, idx)

                save_progress(idx)  # ذخیره وضعیت پردازش

                idx += 1
                success_category = True

            except (StaleElementReferenceException, TimeoutException)
as e:
                retry_category += 1
                print(f"خطا {e} در دسته {idx + 1}، تلاش مجدد
{retry_category}/{max_retry_category}")
                time.sleep(2)
            except Exception as e:
                print(f"خطا در پردازش دسته {idx + 1}: {e}")
                idx += 1
                success_category = True

        if not success_category:
            print(f"عدم موفقیت در پردازش دسته {idx + 1}، پس از چند تلاش، ادامه به دسته
بعدی.")
            idx += 1
```

.1شروع تابع:

def process_all_categories(driver, wait):

- def

برای تعریف یک تابع در پایتون استفاده می‌شود.

- process_all_categories

اسم تابع که به وضوح نشان می‌دهد که این تابع مسئول پردازش تمام دسته‌ها در یک صفحه است.

- (driver, wait)

driver: شیء مرورگر برای کنترل مرورگر.

wait: شیء WebDriverWait که به ما این امکان را می‌دهد که تا زمانی که یک عنصر در صفحه قابل مشاهده یا قابل تعامل باشد، منتظر بمانیم.

.2منتظر بودن تا صفحه بارگذاری شود : wait.until(lambda d: d.execute_script("return document.readyState") == "complete")

- wait.until(lambda d: d.execute_script("return document.readyState") == "complete")

این خط منتظر می‌ماند تا صفحه وب به طور کامل بارگذاری شود. با استفاده از جاوااسکریپت وضعیت بارگذاری (readyState) را بررسی می‌کند.

.3اسکرول به وسط صفحه:

simple_scroll_to_middle(driver)

print("اسکرول به وسط صفحه انجام شد").

◆ simple_scroll_to_middle(driver)

این تابع (که قبلاً تعریف شده) صفحه را به وسط اسکرول می‌کند تا مطمئن شویم که همه دسته‌ها بارگذاری شده‌اند.

◆ print("اسکرول به وسط صفحه انجام شد").

پیامی برای اطلاع از اینکه اسکرول به وسط صفحه انجام شده است.

4.استخراج تمام دسته‌ها:

```
categories = wait.until(EC.presence_of_all_elements_located((By.CLASS_NAME, "callery_posts_box")))
```

◆ categories = wait.until(EC.presence_of_all_elements_located((By.CLASS_NAME, "callery_posts_box")))

منتظر می‌مانیم تا تمام دسته‌ها با کلاس callery_posts_boxدر صفحه حاضر شوند.

categoriesلیستی از تمام این دسته‌ها را ذخیره می‌کند.

5.بارگذاری وضعیت پردازش قبلی:

```
idx = load_progress()  # بارگذاری وضعیت پردازش از فایل
```

◆ idx = load_progress()

وضعیت پردازش قبلی از فایل (که در تابع save_progressذخیره شده است) بارگذاری می‌شود idx . نشان‌دهنده شاخص دسته‌ای است که قبلاً پردازش شده است.

6.تنظیم تعداد تلاش‌های مجدد:

```
max_retry_category = 3
```

◆ max_retry_category = 3

این مقدار حداکثر تعداد تلاش‌ها برای پردازش یک دسته را تعیین می‌کند. اگر بعد از ۳ تلاش موفق نشد، به دسته بعدی می‌رویم.

.7حلقه اصلی برای پردازش دسته‌ها:

while idx < len(categories):

retry_category = 0

success_category = False

◆ while idx < len(categories):

این حلقه برای پیمایش تمام دسته‌ها است. ادامه پیدا می‌کند تا زمانی که idxکمتر از تعداد دسته‌ها باشد.

◆ retry_category = 0

برای هر دسته، تعداد تلاش‌ها را از صفر شروع می‌کنیم.

◆ success_category = False

متغیر success_categoryبرای پیگیری موفقیت‌آمیز بودن پردازش دسته است. در ابتدا مقدار آن Falseاست.

.8تلاش برای پردازش هر دسته:

while retry_category < max_retry_category and not success_category:

try:

◆ while retry_category < max_retry_category and not success_category:

این حلقه داخلی برای تلاش مجدد است. اگر پردازش دسته‌ای ناموفق بود و تعداد تلاش‌ها کمتر از max_retry_categoryباشد، دوباره تلاش می‌کند.

◆ try:

این بخش تلاش می‌کند که کد اجرا شود. اگر خطا رخ دهد، وارد بخش exceptمی‌شود.

9.استخراج و کلیک روی دسته:

```
categories = wait.until(EC.presence_of_all_elements_located((By.CLASS_NAME, "callery_posts_box")))

if idx >= len(categories):
    print("دسته‌ای باقی نمانده یا به انتها رسیدیم. پایان برنامه(".")
    return

category_box = categories[idx]
category_name = category_box.find_element(By.CLASS_NAME, "callery_posts_box_title").text.strip()

driver.execute_script("""
arguments[0].scrollIntoView({behavior: 'smooth', block: 'center', inline: 'nearest'});
""", category_box)
time.sleep(0.5)

try:
    category_box.click()
except ElementClickInterceptedException:
    driver.execute_script("window.scrollTo(0, 0);")
    time.sleep(0.5)
```

category_box.click()

🔷 category_box = categories[idx]

دسته فعلی که در حال پردازش است، استخراج می‌شود.

🔷 category_name = category_box.find_element(By.CLASS_NAME, "callery_posts_box_title").text.strip()

نام دسته استخراج می‌شود.

🔷 driver.execute_script(...)

اسکرول به سمت دسته برای اطمینان از اینکه دسته در دید کاربر قرار گیرد.

🔷 category_box.click()

کلیک روی دسته برای باز کردن یا انجام هر عملیات دیگر.

🔷 except ElementClickInterceptedException:

اگر کلیک اولیه موفق نباشد) مثلاً به دلیل مشکلاتی در DOM) ، از جاوااسکریپت برای کلیک استفاده می‌کنیم.

10.انتظار و استخراج زیر دسته‌ها:

print(f"کلیک روی دسته {idx + 1} انجام شد{category_name}" :

wait.until(EC.presence_of_element_located((By.CLASS_NAME, "callery_product")))

all_results = click_all_subcategories_and_extract(driver, wait, category_box, idx, category_name)

🔷 wait.until(EC.presence_of_element_located((By.CLASS_NAME, "callery_product")))

منتظر می‌مانیم تا زیر دسته‌ها با کلاس callery_productدر دسته بارگذاری شوند.

◆ all_results = click_all_subcategories_and_extract(...)

تابع click_all_subcategories_and_extractبرای کلیک روی تمام زیر دسته‌ها و استخراج داده‌ها فراخوانی می‌شود.

11.ذخیره‌سازی داده‌ها و وضعیت:

save_category_data_to_excel(all_results, category_name, idx)

save_progress(idx) # ذخیره وضعیت پردازش

idx += 1

success_category = True

◆ save_category_data_to_excel(all_results, category_name, idx)

داده‌های استخراج شده برای این دسته در فایل Excel ذخیره می‌شوند.

◆ save_progress(idx)

وضعیت پردازش ذخیره می‌شود تا در صورتی که پردازش متوقف شود، از آن وضعیت بتوانیم ادامه دهیم.

◆ idx += 1

شاخص دسته را افزایش می‌دهیم تا دسته بعدی پردازش شود.

12.مدیریت استثناها:

except (StaleElementReferenceException, TimeoutException) as e:

retry_category += 1

print(f"خطا {e} در دسته{idx + 1} ، تلاش مجدد {retry_category}/{max_retry_category}")

time.sleep(2)

except Exception as e:

print(f"خطا در پردازش دسته{idx + 1}: {e}")

idx += 1

success_category = True

◆ except (StaleElementReferenceException, TimeoutException) as e:

این بخش برای مدیریت استثناهایی مانند مشکلات در تعامل با عناصر DOM و زمان‌بندی (زمانی که صفحه بارگذاری نمی‌شود) استفاده می‌شود.

◆ except Exception as e:

برای مدیریت خطاهای غیرمنتظره در پردازش هر دسته.

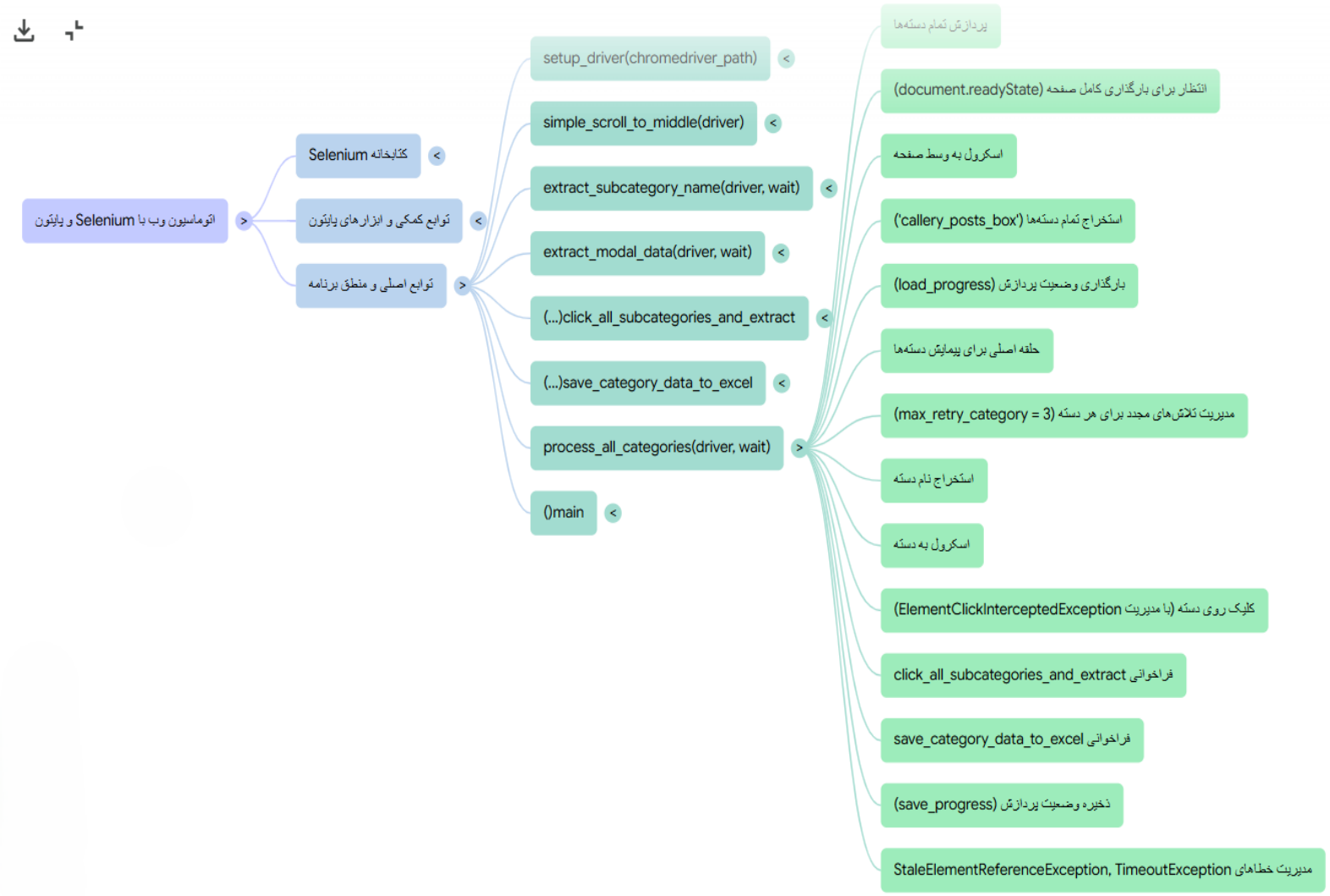
13.پایان تلاش‌های ناموفق:

if not success_category:

print(f"عدم موفقیت در پردازش دسته {idx + 1} پس از چند تلاش، ادامه به دسته بعدی".)

idx += 1

◆اگر پس از چند تلاش (max_retry_category) پردازش یک دسته موفق نشد، پیامی چاپ می‌شود و به دسته بعدی می‌رویم.




```python
# اجرای کد
def main():
    chromedriver_path = "F:\\project_Calori\\chromedriver.exe"  # مسیر کروم‌درایور را اینجا تنظیم کن
    driver = setup_driver(chromedriver_path)
    wait = WebDriverWait(driver, 20)

    url = "https://badankhooba.com/calculating-calories/"
    driver.get(url)
```

```
    process_all_categories(driver, wait)

    driver.quit()

if __name__ == "__main__":
    main()
```

1.تعریف تابع main:

def main():

🔷 def main():

این بخش، شروع تعریف تابع اصلی mainاست که مسئول اجرای تمام مراحل برنامه است.

2.تعریف مسیر کروم‌درایور:

chromedriver_path = "F:\\project_Calori\\chromedriver.exe" # مسیر کروم‌درایور را اینجا تنظیم کن

🔷 chromedriver_path = "F:\\project_Calori\\chromedriver.exe"

مسیر فایل chromedriver.exeرا در این خط مشخص می‌کنیم. این فایل برای تعامل بین Selenium و مرورگر کروم استفاده می‌شود.

مسیر به شکل محلی درایو Fقرار داده شده است.

3.راه‌اندازی مرورگر:

driver = setup_driver(chromedriver_path)

🔷 driver = setup_driver(chromedriver_path)

در این خط، تابع setup_driverفراخوانی می‌شود تا مرورگر کروم را با استفاده از کروم‌درایور راه‌اندازی کند.

chromedriver_pathبه این تابع به عنوان ورودی داده می‌شود تا مرورگر کروم را با مسیر مشخص شده راه‌اندازی کند.

4.ایجاد شیء :WebDriverWait

```
wait = WebDriverWait(driver, 20)
```

◆ wait = WebDriverWait(driver, 20)

WebDriverWaitیک شیء است که به ما این امکان را می‌دهد تا مدت زمانی مشخص (در اینجا ۲۰ ثانیه) منتظر بمانیم تا یک عنصر در صفحه بارگذاری و قابل دسترسی باشد.

5.بارگذاری صفحه وب:

```
url = "https://badankhooba.com/calculating-calories/"
driver.get(url)
```

◆ url = "https://badankhooba.com/calculating-calories/"

در این خط، آدرس صفحه‌ای که باید باز شود را ذخیره می‌کنیم.

◆ driver.get(url)

از این متد برای بارگذاری صفحه با آدرس مشخص شده استفاده می‌کنیم.

6.پردازش تمام دسته‌ها:

```
process_all_categories(driver, wait)
```

◆ process_all_categories(driver, wait)

این خط تابع process_all_categoriesرا فراخوانی می‌کند تا تمام دسته‌ها در صفحه پردازش شوند. این تابع دسته‌ها را شناسایی، کلیک و داده‌های مربوطه را استخراج می‌کند.

7.بستن مرورگر:

driver.quit()

🔷 driver.quit()

این متد برای بستن مرورگر و تمام پروسه‌های مربوط به Selenium استفاده می‌شود.

8.بررسی اجرای اسکریپت:

if __name__ == "__main__":

main()

🔷 if __name__ == "__main__":

این خط بررسی می‌کند که آیا اسکریپت به طور مستقیم اجرا شده است یا خیر. اگر اسکریپت مستقیماً اجرا شده باشد، تابع main()فراخوانی می‌شود.