Submitted to:

TATA CONSULTANCY SERVICES

# Automated Price Tracker Bot

Author:

Sagnik Mitra[a], Shivani Dubey[b]

[a][b] Department of Computer Science & Business Systems, Sister Nivedita University

Project Mentor:

S CH H Reddy Mallidi

Tata Consultancy Services

## 1 ABSTRACT

In the world of Online Shopping and E-Commerce, website we often don't find the time to compare prices or search some products. It would always be very helpful if we had some Price Tracker that would compare the prices and would return the product details with an integrated automation so that we don't need to do it manually. Price History for over 1000 million Amazon products, even for different versions, colors or sizes of a product. Everything is at one click from you.

## 2 INTRODUCTION

In this Project, we have tried to work on a Python Automated Bot for Price Tracking that would help the user to Track the price of any product that is there on Amazon. This is implemented for Amazon. but just by changing the URLs and few settings it can be used for any of the E-Commerce website that are available on the internet and also we have tried to initiate the price comparison for the product and also added the functionalities to fetch the product according to their review. The bot has an integration of filter based search. The user has the option to search based on Name, Base_URL, Price, Currency. In the subsequent part of the Project Report we have discussed the implementation process.

### 2.1 Application

The Bot or the Script can be used to track price of any product from any of amazon's categories. But also, with few tweaks, we can make it happen for any of the available E-Commerce websites. This will help the user to search for any product without hassle of going to Amazon's website and manually searching and finding the product. They can just simple run the script and do the same.

## 3 Test Data

```
test_data = [{'asin': 'B07WHSY2WT', 'url': 'http://www.amazon.de/dp/B07WHSY2WT',
              'title': 'PlayStation 4 Slim - Konsole (1 TB, schwarz) inkl. FIFA 20 + 2 DualShock Controller',
              'seller': 'Sony Interactive Entertainment', 'price': 379.0},
             {'asin': 'B085PB4B6J', 'url': 'http://www.amazon.de/dp/B085PB4B6J',
              'title': 'PlayStation 4 Pro - Konsole (1 TB, schwarz) PS Hits Naughty Dog Bundle',
              'seller': 'Sony Interactive Entertainment', 'price': 399.99},
             {'asin': 'B07HHPX4N1', 'url': 'http://www.amazon.de/dp/B07HHPX4N1',
              'title': 'PlayStation 4 - Konsole (500 GB, schwarz, slim, F-Chassis) inkl. 2 DualShock 4 Controller',
              'seller': 'Sony Interactive Entertainment', 'price': 299.99},
             {'asin': 'B07HSJW7HK', 'url': 'http://www.amazon.de/dp/B07HSJW7HK',
              'title': 'PlayStation 4 Pro - Konsole (1 TB, schwarz, Pro, Modell: CUH-7216B)',
              'seller': 'Sony Interactive Entertainment', 'price': 408.0},
             {'asin': 'B07KMV94JF', 'url': 'http://www.amazon.de/dp/B07KMV94JF',
              'title': 'PlayStation 4 Slim - Konsole (1TB, schwarz)', 'seller': 'Sony Interactive Entertainment',
              'price': 359.0}, {'asin': 'B07HNR4ZZD', 'url': 'http://www.amazon.de/dp/B07HNR4ZZD',
                                'title': 'PlayStation4 - Konsole (500GB, schwarz, slim)', 'seller': 'Sony',
                                'price': 317.0}, {'asin': 'B07V7NT6ZC', 'url': 'http://www.amazon.de/dp/B07V7NT6ZC',
                                                  'title': 'PlayStation 4 Pro (1TB, black): Fortnite Neo Versa Bundle',
                                                  'seller': 'Sony', 'price': 409.98},
             {'asin': 'B07V282WBB', 'url': 'http://www.amazon.de/dp/B07V282WBB',
```

```
21              'title': 'PlayStation 4 Slim - Konsole (500GB, Jet Black) +
     2 Controller: Fortnite Neo Versa Bundle',
22              'seller': 'Sony', 'price': 369.0}, {'asin': 'B07YGJGFZC', '
    url': 'http://www.amazon.de/dp/B07YGJGFZC',
23                                                  'title': 'PlayStation 4
     Virtual Reality Megapack - Edition 2 (inkl. Skyrim, Astro Bot Rescue
    Mission, VR Worlds, Resident Evil: Biohazard, Everybody s Golf)',
24                                                  'seller': 'Sony
    Interactive Entertainment', 'price': 379.0},
25             {'asin': 'B07K2PT733', 'url': 'http://www.amazon.de/dp/
    B07K2PT733', 'title': 'PlayStation VR',
26              'seller': 'Sony Interactive Entertainment', 'price':
    289.0},
27             {'asin': 'B07Z7438CY', 'url': 'http://www.amazon.de/dp/
    B07Z7438CY',
28              'title': 'Spongebob SquarePants: Battle for Bikini Bottom -
     Rehydrated - F.U.N. Edition [Playstation 4]',
29              'seller': 'THQ Nordic', 'price': 299.99},
30             {'asin': 'B07DNM3MT9', 'url': 'http://www.amazon.de/dp/
    B07DNM3MT9',
31              'title': 'PlayStation 4 Pro - Konsole Schwarz, A Chassis, 1
    TB, (General berholt)',
32              'seller': 'Sony Interactive Entertainment', 'price':
    349.99},
33             {'asin': 'B07WDKT7DP', 'url': 'http://www.amazon.de/dp/
    B07WDKT7DP',
34              'title': 'PlayStation 4 Pro - Konsole (1TB) inkl. FIFA 20',
35              'seller': 'Sony Interactive Entertainment', 'price':
    419.0},
36             {'asin': 'B07YSX4DLW', 'url': 'http://www.amazon.de/dp/
    B07YSX4DLW',
37              'title': 'PlayStation 4 Slim inkl. 2 Controller und Call of
     Duty: Modern Warfare - Konsolenbundle (1TB, schwarz, Slim)',
38              'seller': 'Sony', 'price': 444.0}, {'asin': 'B07S7RGRFC', '
    url': 'http://www.amazon.de/dp/B07S7RGRFC',
39                                                  'title': 'Playstation 4
     Slim 1TB - Limited Edition Days of Play 2019',
40                                                  'seller': 'Sony', '
    price': 359.0},
41              ]
```

# 4 Libraies

## 4.1 Used Libraies

**selenium:** Selenium is a portable framework for testing web applications. Selenium provides a playback tool for authoring functional tests without the need to learn a test scripting language. .

**amazon_config:** Amazon config is a service that enables you to assess, audit, and evaluate the configurations of your Amazon Web Services resources. Config continuously monitors and records your Amazon Web Services resource configurations and allows you to automate

the evaluation of recorded configurations against desired configurations.

**json:** JSON (JavaScript Object Notation) is a lightweight data-interchange format.It is easy for humans to read and write. It is easy for machines to parse and generate.

**bs4:** Beautiful Soup is a Python library for pulling data out of HTML and XML files. It works with your favorite parser to provide idiomatic ways of navigating, searching, and modifying the parse tree.

**pymongo:** PyMongo is a Python distribution containing tools for working with MongoDB, and is the recommended way to work with MongoDB from Python.

**python-dateutil:** The dateutil module provides powerful extensions to the standard date-time module, available in Python.

**tqdm:** tqdm instantly make your loops show a smart progress meter.It works on any platform.

**urllib3:** urllib3 is a powerful, user-friendly HTTP client for Python.It brings many critical features that are missing from the Python standard libraries such as thread safety, connection pooling etc.

**mongoengine:** MongoEngine is a Python library that acts as an Object Document Mapper with MongoDB, a NOSQL database. It is similar to SQLAlchemy, which is the Object Relation Mapper (ORM) for SQL based databases.

**idna:** This is a library to support the Internationalised Domain Names in Applications (IDNA) protocol as specified in RFC 5891. This version of the protocol is often referred to as "IDNA2008" and can produce different results from the earlier standard from 2003. The library is also intended to act as a suitable drop-in replacement for the "encodings.idna" module that comes with the Python standard library but currently only supports the older 2003 specification.

**chardet:** Character encoding auto-detection in Python. As smart as your browser. Open source.

**certifi:** Certifi provides Mozilla's carefully curated collection of Root Certificates for validating the trustworthiness of SSL certificates while verifying the identity of TLS hosts. It has been extracted from the Requests project.

## 4.2 Version Requirements

```
beautifulsoup4==4.9.0\\
bs4==0.0.1\\
certifi==2020.4.5.1\\
chardet==3.0.4\\
colorama==0.4.3\\
```

```
idna==2.9\\
mongoengine==0.19.1\\
pymongo==3.10.1\\
python-dateutil==2.8.1\\
requests==2.23.0\\
selenium==3.141.0\\
six==1.14.0\\
soupsieve==2.0\\
tqdm==4.45.0\\
urllib3==1.25.9\\
```

# 5 Experimental Results

Fig. 1 Growth of Error Rate with respect to K

# 6 Configuration

At first we need to import the webdriver and implement the web_driver functions to successfully scrap thought the web.

## 6.1 Setting up the Base Details

```
from selenium import webdriver

DIRECTORY = 'reports'
NAME = 'PS4'
CURRENCY = '   '
MIN_PRICE = '275'
MAX_PRICE = '650'
FILTERS = {
    'min': MIN_PRICE,
    'max': MAX_PRICE
}
BASE_URL = "http://www.amazon.de/"
```

## 6.2 Browser Driver Libraries

```
def get_chrome_web_driver(options):
    return webdriver.Chrome("./chromedriver", chrome_options=options)


def get_web_driver_options():
    return webdriver.ChromeOptions()


def set_ignore_certificate_error(options):
    options.add_argument('--ignore-certificate-errors')

```

```
12
13 def set_browser_as_incognito(options):
14     options.add_argument('--incognito')
15
16
17 def set_automation_as_head_less(options):
18     options.add_argument('--headless')
```

# 7 Tracker

## 7.1 Library Import

Importing the necessary libraries and fetching the necessary driver functions from the amazon_config file mentioned in the previous section.

```
1  import time
2  from selenium.webdriver.common.keys import Keys
3  from amazon_config import (
4      get_web_driver_options,
5      get_chrome_web_driver,
6      set_ignore_certificate_error,
7      set_browser_as_incognito,
8      set_automation_as_head_less,
9      NAME,
10     CURRENCY,
11     FILTERS,
12     BASE_URL,
13     DIRECTORY
14 )
15 from selenium.common.exceptions import NoSuchElementException
16 import json
17 from datetime import datetime
```

## 7.2 Product Report Generation

Fetching the real time price according to the urls taken in the test data.

```
1  class GenerateReport:
2      def __init__(self, file_name, filters, base_link, currency, data):
3          self.data = data
4          self.file_name = file_name
5          self.filters = filters
6          self.base_link = base_link
7          self.currency = currency
8          report = {
9              'title': self.file_name,
10             'date': self.get_now(),
```

```
            'best_item': self.get_best_item(),
            'currency': self.currency,
            'filters': self.filters,
            'base_link': self.base_link,
            'products': self.data
        }
        print("Creating report...")
        with open(f'{DIRECTORY}/{file_name}.json', 'w') as f:
            json.dump(report, f)
        print("Done...")

    @staticmethod
    def get_now():
        now = datetime.now()
        return now.strftime("%d/%m/%Y %H:%M:%S")

    def get_best_item(self):
        try:
            return sorted(self.data, key=lambda k: k['price'])[0]
        except Exception as e:
            print(e)
            print("Problem with sorting items")
            return None
```

## 7.3 The API Class

### 7.3.1 Initialization

Initializing the Amazon API class with the following attributes.

- **Base Url** for fetching the Product Link from Amazon
- **Search Term** based on which we will filter the data
- **Options** from the web driver
- **Chrome Web Driver** for setting up the driver
- **Currency** for setting up the Setting the Currency
- **Price Filter** to get customized data

```
class AmazonAPI:
    def __init__(self, search_term, filters, base_url, currency):
        self.base_url = base_url
        self.search_term = search_term
        options = get_web_driver_options()
        # set_automation_as_head_less(options)
        set_ignore_certificate_error(options)
        set_browser_as_incognito(options)
        self.driver = get_chrome_web_driver(options)
        self.currency = currency
        self.price_filter = f"&rh=p_36%3A{filters['min']}00-{filters['max
    ']}00"
```

### 7.3.2 Starting the Web Script

Starting the Web Script and searching the item over the web. Getting info about the product along with the product links.

```
def run(self):
        print("Starting Script...")
        print(f"Looking for {self.search_term} products...")
        links = self.get_products_links()
        if not links:
            print("Stopped script.")
            return
        print(f"Got {len(links)} links to products...")
        print("Getting info about products...")
        products = self.get_products_info(links)
        print(f"Got info about {len(products)} products...")
        self.driver.quit()
        return products

def get_products_links(self):
        self.driver.get(self.base_url)
        element = self.driver.find_element_by_xpath(
            '//*[@id="twotabsearchtextbox"]')
        element.send_keys(self.search_term)
        element.send_keys(Keys.ENTER)
        time.sleep(2)  # wait to load page
        self.driver.get(f'{self.driver.current_url}{self.price_filter}')
        print(f"Our url: {self.driver.current_url}")
        time.sleep(2)  # wait to load page
        result_list = self.driver.find_elements_by_class_name('s-result-
    list')
        links = []
        try:
            results = result_list[0].find_elements_by_xpath(
                "//div/span/div/div/div[2]/div[2]/div/div[1]/div/div/div
    [1]/h2/a")
            links = [link.get_attribute('href') for link in results]
            return links
        except Exception as e:
            print("Didn't get any products...")
            print(e)
            return links

def get_products_info(self, links):
        asins = self.get_asins(links)
        products = []
        for asin in asins:
            product = self.get_single_product_info(asin)
            if product:
                products.append(product)
        return products
```

### 7.3.3 Single Product Info

```
def get_asins(self, links):
        return [self.get_asin(link) for link in links]

    def get_single_product_info(self, asin):
        print(f"Product ID: {asin} - getting data...")
        product_short_url = self.shorten_url(asin)
        self.driver.get(f'{product_short_url}?language=en_GB')
        time.sleep(2)
        title = self.get_title()
        seller = self.get_seller()
        price = self.get_price()
        if title and seller and price:
            product_info = {
                'asin': asin,
                'url': product_short_url,
                'title': title,
                'seller': seller,
                'price': price
            }
            return product_info
        return None
```

### 7.3.4 Get Title Seller Info

```
def get_title(self):
        try:
            return self.driver.find_element_by_id('productTitle').text
        except Exception as e:
            print(e)
            print(f"Can't get title of a product - {self.driver.
    current_url}")
            return None

    def get_seller(self):
        try:
            return self.driver.find_element_by_id('bylineInfo').text
        except Exception as e:
            print(e)
            print(f"Can't get seller of a product - {self.driver.
    current_url}")
            return None
```

### 7.3.5 Get Product Price

```
def get_price(self):
        price = None
        try:
            price = self.driver.find_element_by_id('priceblock_ourprice')
    .text
```

```
            price = self.convert_price(price)
        except NoSuchElementException:
            try:
                availability = self.driver.find_element_by_id(
                    'availability').text
                if 'Available' in availability:
                    price = self.driver.find_element_by_class_name(
                        'olp-padding-right').text
                    price = price[price.find(self.currency):]
                    price = self.convert_price(price)
            except Exception as e:
                print(e)
                print(
                    f"Can't get price of a product - {self.driver.
    current_url}")
                return None
        except Exception as e:
            print(e)
            print(f"Can't get price of a product - {self.driver.
    current_url}")
            return None
        return price
```

### 7.3.6 Shortening URLs

Setting the methods as Static

```
@staticmethod
    def get_asin(product_link):
        return product_link[product_link.find('/dp/') + 4:product_link.
    find('/ref')]

    def shorten_url(self, asin):
        return self.base_url + 'dp/' + asin
```

### 7.3.7 Convert Price

Splitting the Price String to convert the Price into Float DataType

```
def convert_price(self, price):
        price = price.split(self.currency)[1]
        try:
            price = price.split("\n")[0] + "." + price.split("\n")[1]
        except:
            Exception()
        try:
            price = price.split(",")[0] + price.split(",")[1]
        except:
            Exception()
        return float(price)
```

#### 7.3.8 Driver Function

```
if __name__ == '__main__':
    am = AmazonAPI(NAME, FILTERS, BASE_URL, CURRENCY)
    data = am.run()
    GenerateReport(NAME, FILTERS, BASE_URL, CURRENCY, data)
```

## 8 Source Code

**GitHub Source Code:** https://github.com/sagnikmitra/product-bot

## 9 Conclusion

In this Project, we have tried to use the Chrome Web Driver to automate the process of searching a Product over the internet and while searching it, we have scraped the product data through our script and returned in our local machine where the user will be able to see without any hassle or even without going to the amazon site, just by searching on out platform. We have implemented the backend using Python, Scraped the Web Console data using Beautiful Soap and Automated the Process using Selenium Web Engine. This will help to get data from different e-commerece website by some simple tweaks in the program and test data. The data is based on Deutsch Amazon site as the Indian site was not stable while we were scraping the data and it showed fluctuation error. That's why we took this test dataset and worked accrodingly.

### 9.1 Future Work

We will try to build a React Portal that would help in interacting with the Script on the Frontend part rather than manually running everytime. Also, due to fluctuation, error, we couldn't take the Indian Amazon Site data and had to fetch from the deutsch Amazon Data. In the future, we will try to preprocess the data in a different way such that any country sites can be scraped and implemented for the process. There is also a scope of making the user choose between several different E-Commerce Sites

## 10 Bibliography

### 10.1 References

**[1] Matrin RiedWeg Zentaon Puppeter**
**[2] DB Implementation of Price Tracker for E-Commerce**