

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра Автоматики и Процессов Управления**

**ОТЧЕТ**  
**по «Автоматизация тестирования» практике**  
**Тема: Тестирование с помощью Selenium webdriver и Python**

Студент гр. 2307

Подберёзский А. Д.

Руководитель

Турнецкая Е.Л.

Санкт-Петербург

2024

## ЗАДАНИЕ

### НА «АВТОМАТИЗАЦИЯ ТЕСТИРОВАНИЯ» ПРАКТИКУ

Студент Подберёзский А.Д.

Группа 2307

Тема практики: Тестирование с помощью Selenium webdriver и Python

Задание на практику:

Получение практических навыков по тестированию веб-элементов с помощью Selenium Webdriver и Python.

Для достижения поставленной цели требуется решить следующие задачи:

1. Установить программное окружение проекта по автоматизированному тестированию.
2. Реализовать проект по запуску браузера под управлением Selenium и отобразить в нем страницу веб-системы.
3. Разработать проект по нахождению веб-элементов на учебном ресурсе с помощью программных методов Selenium.
4. Создать проект по заполнению полей формы регистрации с использованием явных и неявных ожиданий Selenium.
5. Зафиксировать результаты тестирования в отчете

Сроки прохождения практики: 02.09.2024 – 21.12.2024

Дата сдачи отчета: 10.10.2024

Дата защиты отчета: 10.10.2024

Студент		Подберёзский А.Д.
Руководитель		Турнецкая Е.Л.

## **АННОТАЦИЯ**

В данном отчете рассмотрен процесс создания автоматизированного тестирования веб-элементов с использованием инструмента selenium webdriver и языка программирования python. В рамках практической работы была разработана и протестирована серия скриптов для выполнения различных операций с веб-страницами, включая открытие браузера, поиск элементов на странице, заполнение полей формы, выполнение javascript-кода и фиксацию результатов тестирования. Цель работы — получить практические навыки работы с selenium и применить их для решения задач автоматизации веб-тестирования. Полученные результаты демонстрируют эффективность подхода, а также основные методы взаимодействия с элементами веб-интерфейса.

## **SUMMARY**

This report examines the process of creating automated testing of web elements using the selenium webdriver tool and the python programming language. As part of the practical work, a series of scripts were developed and tested to perform various operations with web pages, including opening the browser, searching for elements on the page, filling in form fields, executing javascript code and fixing test results. The purpose of the work is to gain practical skills in working with selenium and apply them to solving web testing automation tasks. The results obtained demonstrate the effectiveness of the approach, as well as the main methods of interaction with web interface elements.

## **ВВЕДЕНИЕ**

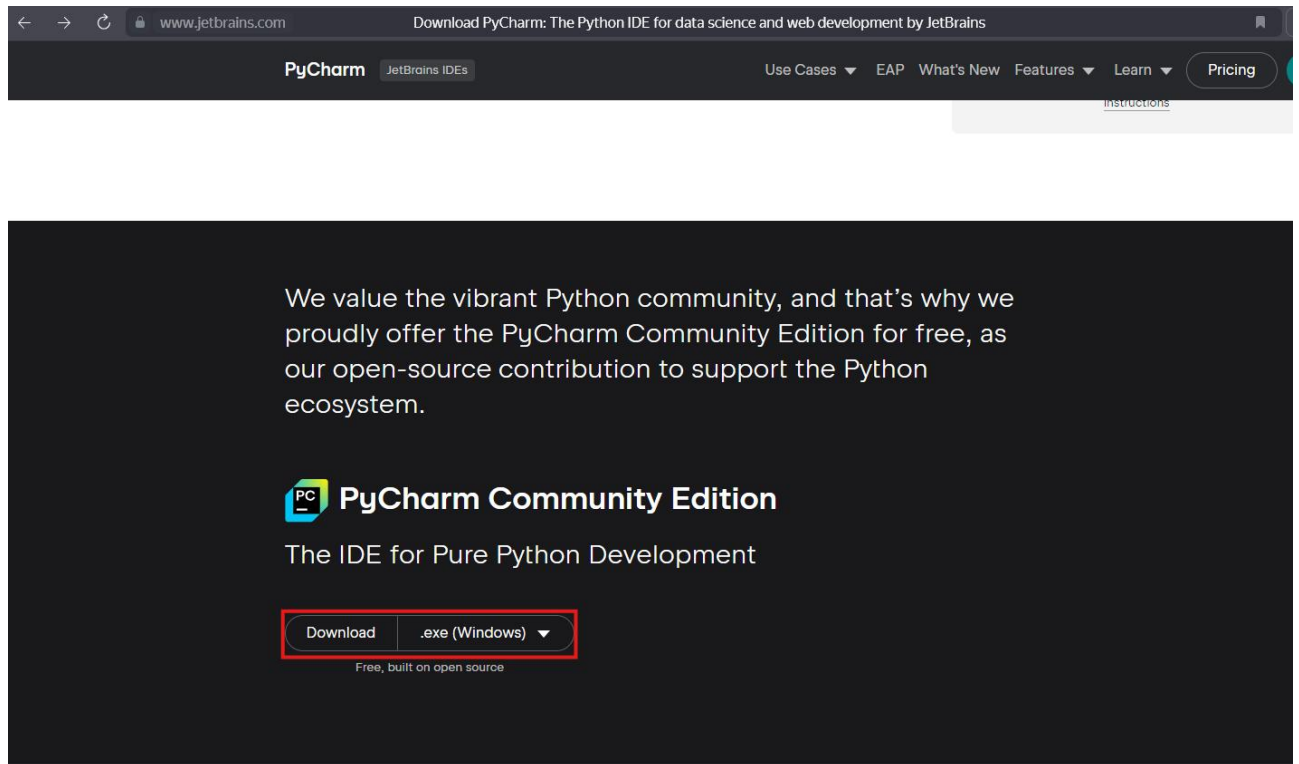
Получение практических навыков по тестированию веб-элементов с помощью Selenium Webdriver и Python.

Для достижения поставленной цели требуется решить следующие задачи:

1. Установить программное окружение проекта по автоматизированному тестированию.
2. Реализовать проект по запуску браузера под управлением Selenium и отобразить в нем страницу веб-системы.
3. Разработать проект по нахождению веб-элементов на учебном ресурсе с помощью программных методов Selenium.
4. Создать проект по заполнению полей формы регистрации с использованием явных и неявных ожиданий Selenium.
5. Зафиксировать результаты тестирования в отчете

## 1. Установка программного окружения

Вначале скачиваем установщик PyCharm Community Edition с официального [сайта](https://www.jetbrains.com/pycharm/) и запускаем его.



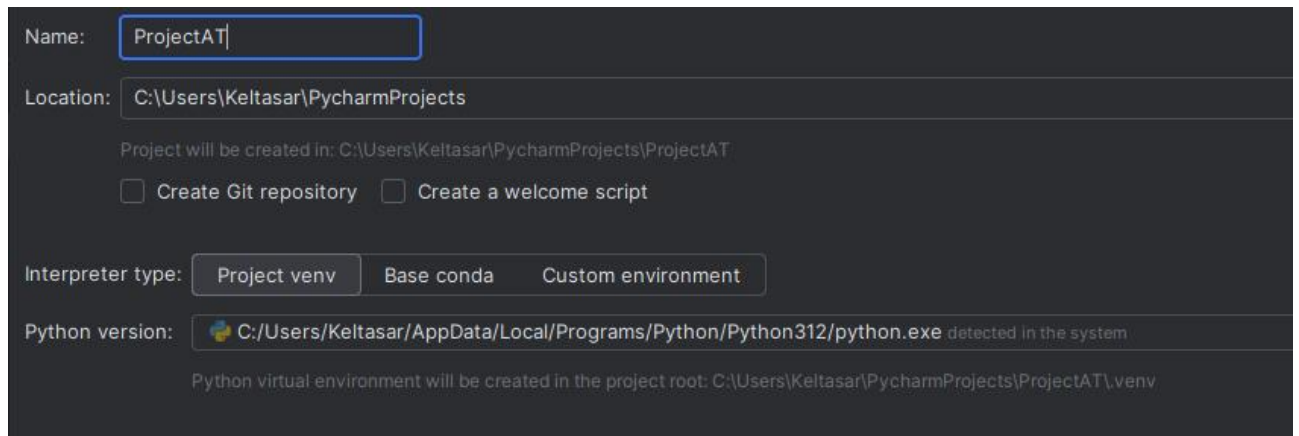
*Рис. 1.1 - Сайт для скачивания PyCharm CE*

В случае успешной установки PyCharm нам предложат его запустить, что мы и сделаем.



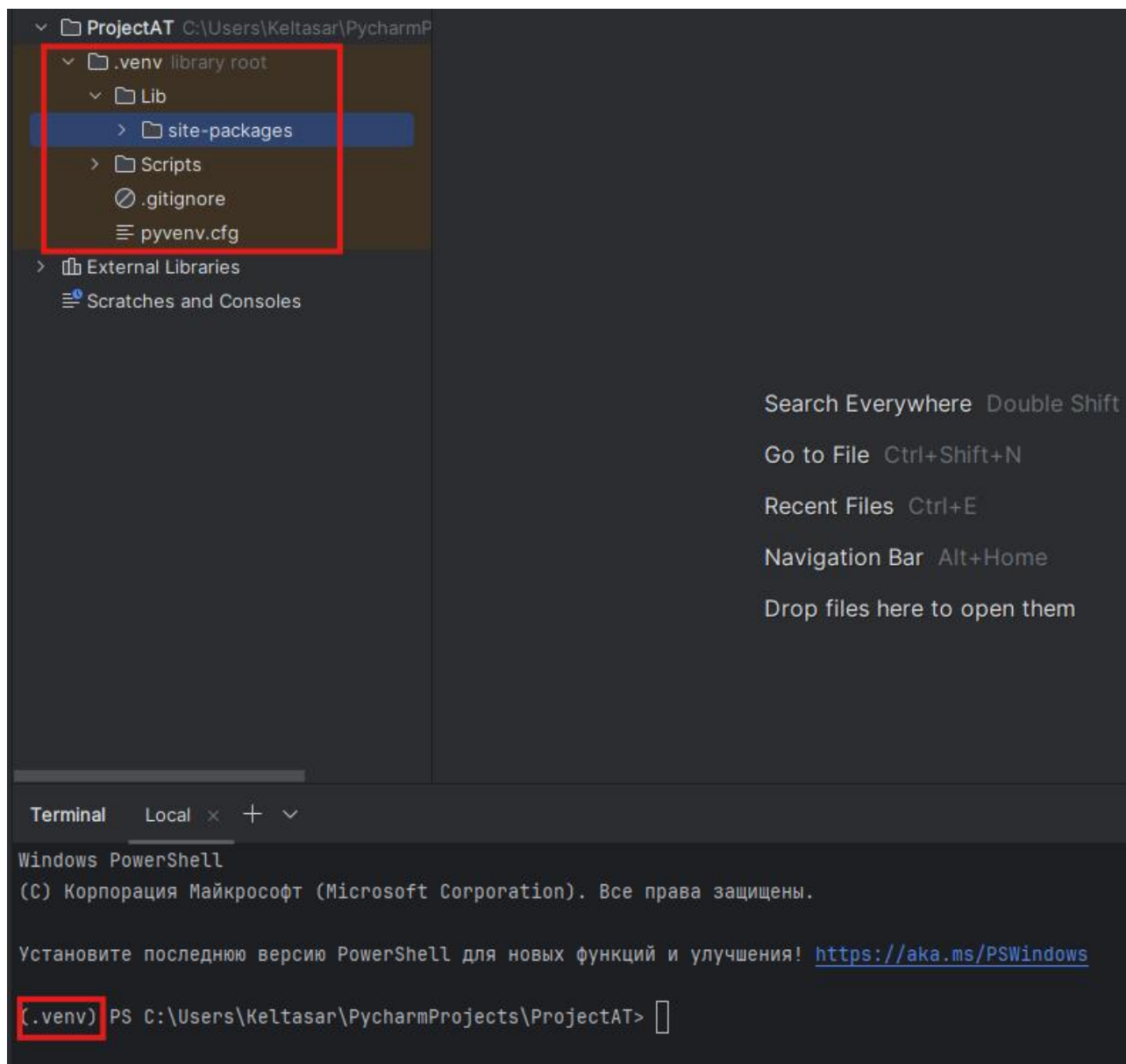
*Рис. 1.2 - Сообщение об успешной установке Pycharm CE*

Далее создаём новый проект и проверяем чтобы в поле “Interpreter type” было выбрано “Project venv”.



*Рис. 1.3 - Первичная настройка проекта*

После создания проекта проверяем наличие виртуального окружения venv.



*Рис. 1.4 - Проверка нахождения виртуального окружения*

Далее проверяю установленные библиотеки командой “pip list”. В случае, если не установлены библиотеки “pytest” и “selenium”, пишем в терминал команду “pip install <имя библиотеки>”.

```
Terminal Local x + v
(.venv) PS C:\Users\Keltasar\PycharmProjects\ProjectAT> pip list
Package            Version
-----
attrs              24.2.0
certifi            2024.8.30
cffi               1.17.1
colorama           0.4.6
h11                0.14.0
idna               3.10
iniconfig          2.0.0
outcome            1.3.0.post0
packaging          24.2
pip               24.3.1
pluggy            1.5.0
pyparser           2.22
PySocks            1.7.1
pytest             8.3.3
selenium           4.26.1
```

Рис. 1.5 - Проверка установки pytest и selenium

После установки PyCharm и настройки виртуального окружения необходимо установить WebDriver. Для этого заходим в браузер, при помощи которого будем тестировать, в моём случае это Google Chrome, и проверяем его версию перейдя в Настройки -> О браузере Chrome.

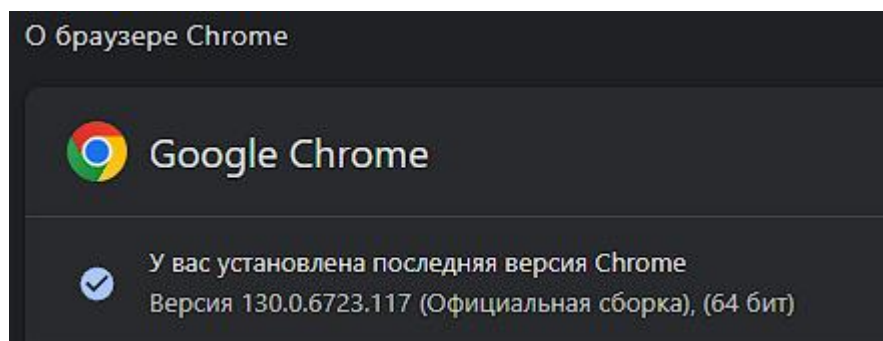


Рис. 1.6 - Проверка версии браузера

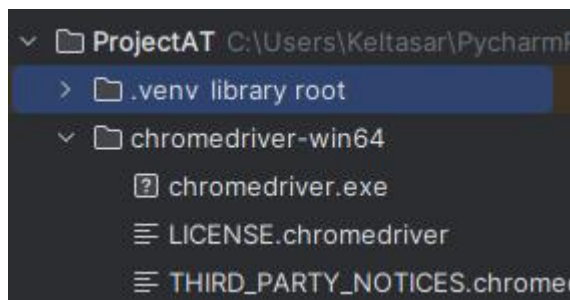
Далее переходим на [сайт](#) и скачиваем webdriver той же версии, что и наш браузер.



Stable		
Version: 130.0.6723.116 (r1356013)		
Binary	Platform	URL
chrome	linux64	<a href="https://storage.googleapis.com/chrome-for-testing-public/130.0.6723.116/linux64/chrome-linux64.zip">https://storage.googleapis.com/chrome-for-testing-public/130.0.6723.116/linux64/chrome-linux64.zip</a>
chrome	mac-arm64	<a href="https://storage.googleapis.com/chrome-for-testing-public/130.0.6723.116/mac-arm64/chrome-mac-arm64.zip">https://storage.googleapis.com/chrome-for-testing-public/130.0.6723.116/mac-arm64/chrome-mac-arm64.zip</a>
chrome	mac-x64	<a href="https://storage.googleapis.com/chrome-for-testing-public/130.0.6723.116/mac-x64/chrome-mac-x64.zip">https://storage.googleapis.com/chrome-for-testing-public/130.0.6723.116/mac-x64/chrome-mac-x64.zip</a>
chrome	win32	<a href="https://storage.googleapis.com/chrome-for-testing-public/130.0.6723.116/win32/chrome-win32.zip">https://storage.googleapis.com/chrome-for-testing-public/130.0.6723.116/win32/chrome-win32.zip</a>
chrome	win64	<a href="https://storage.googleapis.com/chrome-for-testing-public/130.0.6723.116/win64/chrome-win64.zip">https://storage.googleapis.com/chrome-for-testing-public/130.0.6723.116/win64/chrome-win64.zip</a>
chromedriver	linux64	<a href="https://storage.googleapis.com/chrome-for-testing-public/130.0.6723.116/linux64/chromedriver-linux64.zip">https://storage.googleapis.com/chrome-for-testing-public/130.0.6723.116/linux64/chromedriver-linux64.zip</a>
chromedriver	mac-arm64	<a href="https://storage.googleapis.com/chrome-for-testing-public/130.0.6723.116/mac-arm64/chromedriver-mac-arm64.zip">https://storage.googleapis.com/chrome-for-testing-public/130.0.6723.116/mac-arm64/chromedriver-mac-arm64.zip</a>
chromedriver	mac-x64	<a href="https://storage.googleapis.com/chrome-for-testing-public/130.0.6723.116/mac-x64/chromedriver-mac-x64.zip">https://storage.googleapis.com/chrome-for-testing-public/130.0.6723.116/mac-x64/chromedriver-mac-x64.zip</a>
chromedriver	win32	<a href="https://storage.googleapis.com/chrome-for-testing-public/130.0.6723.116/win32/chromedriver-win32.zip">https://storage.googleapis.com/chrome-for-testing-public/130.0.6723.116/win32/chromedriver-win32.zip</a>
chromedriver	win64	<a href="https://storage.googleapis.com/chrome-for-testing-public/130.0.6723.116/win64/chromedriver-win64.zip">https://storage.googleapis.com/chrome-for-testing-public/130.0.6723.116/win64/chromedriver-win64.zip</a>

*Рис. 1.7 - Скачивание WebDriver*

После скачивания zip архива извлекаем всё в директорию проекта.



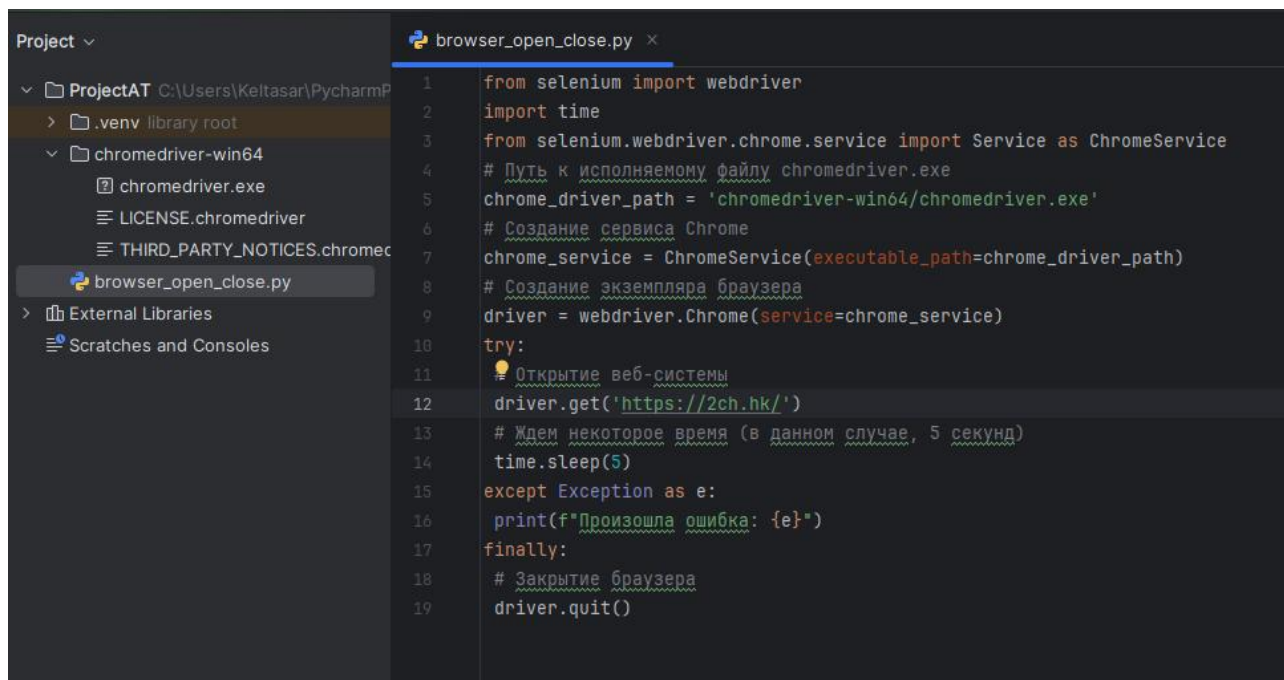
*Рис. 1.8 - Проверка нахождения WebDriver в директории проекта*

На этом этапе настройка программного окружения считается завершённой.

## 2. Выполнение практической части

### 2.1. Открытие веб-страницы

Создаём файл .py с названием “browser\_open\_close.py” и копируем в него код из методических указаний из пункта Листинг 4.1 с заменой адреса на [свой адрес](#).



```
1 from selenium import webdriver
2 import time
3 from selenium.webdriver.chrome.service import Service as ChromeService
4 # Путь к исполняемому файлу chromedriver.exe
5 chrome_driver_path = 'chromedriver-win64/chromedriver.exe'
6 # Создание сервиса Chrome
7 chrome_service = ChromeService(executable_path=chrome_driver_path)
8 # Создание экземпляра браузера
9 driver = webdriver.Chrome(service=chrome_service)
10 try:
11     # Открытие веб-системы
12     driver.get('https://2ch.hk/')
13     # Ждем некоторое время (в данном случае, 5 секунд)
14     time.sleep(5)
15 except Exception as e:
16     print(f"Произошла ошибка: {e}")
17 finally:
18     # Закрытие браузера
19     driver.quit()
```

Рис. 2.1.1 - Код программы для открытия веб-страницы

Запускаем программу и в случае верного выполнения программы введённый сайт открывается в новом окне и выводится надпись «Браузером Chrome управляет автоматизированное тестовое ПО».

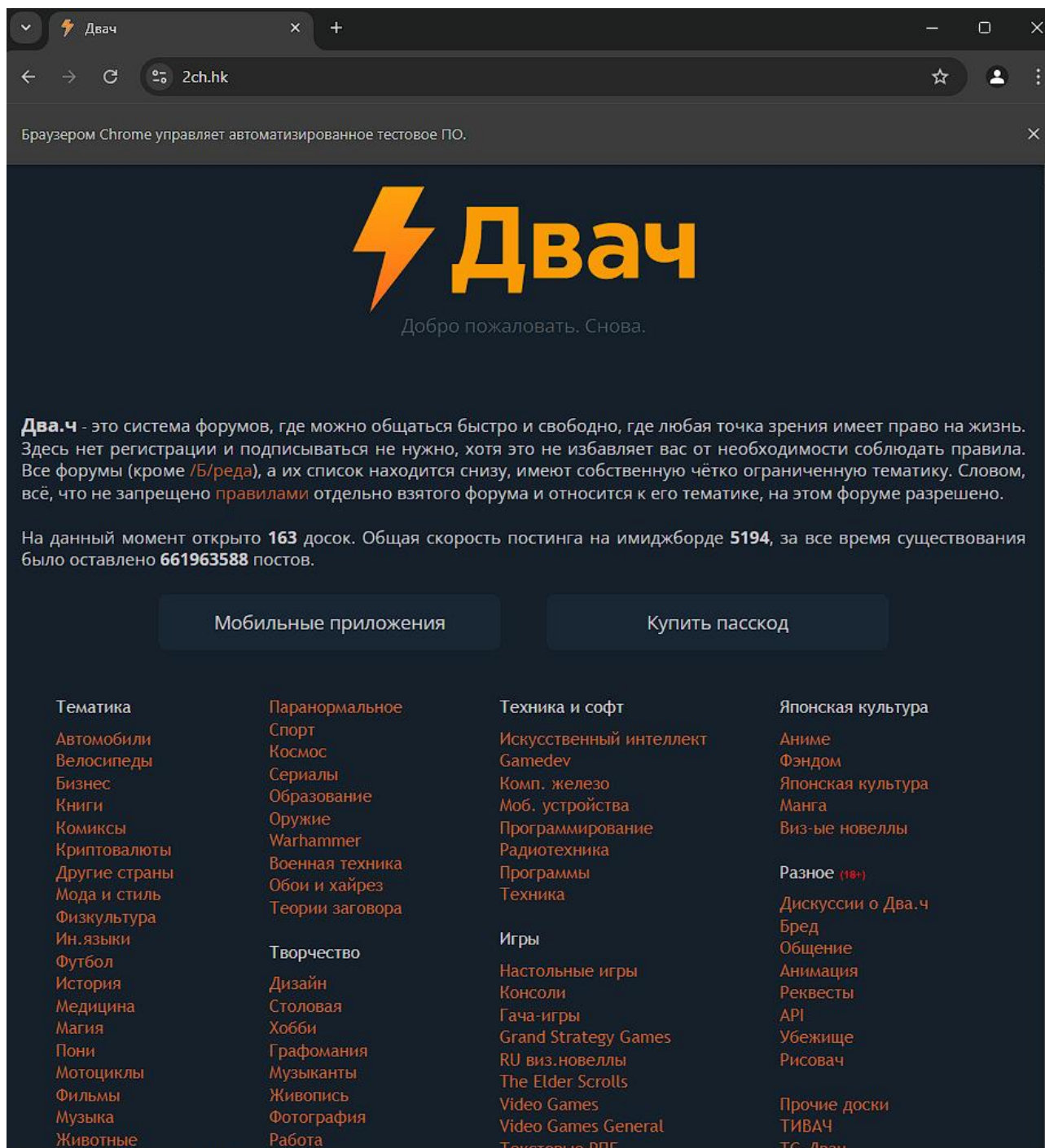


Рис. 2.1.2 - Открытый при помощи программы сайт

## 2.2. Поиск веб-элементов

Для поиска веб-элементов создаём файл “element\_search.py” и копируем в него код из файла “browser\_open\_close.py”. Далее дополняем его для выбора варианта и поиска элементов по заданным параметрам и выводу на экран количество элементов с тегом h1, именем almond-nut, классом imageContainer, ID peanut и атрибутом data-type равным "nuts".

```

from selenium import webdriver
import time
from selenium.webdriver.chrome.service import Service as ChromeService
from selenium.webdriver.common.by import By

# Путь к исполняемому файлу chromedriver.exe
chrome_driver_path = 'chromedriver-win64/chromedriver.exe'
# Создание сервиса Chrome
chrome_service = ChromeService(executable_path=chrome_driver_path)
# Создание экземпляра браузера
driver = webdriver.Chrome(service=chrome_service)
# Ссылка на учебный веб-проект и номер варианта
URL = "https://qa-test-selectors.netlify.app/"
variant = 15
try:
    # Открытие веб-системы
    driver.get(URL)
    # Ожидание загрузки всех элементов страницы
    driver.implicitly_wait(10)
    # Нажатие кнопки варианта
    v_button = driver.find_elements(By.TAG_NAME, value="button")[variant-1]
    v_button.click()
    # Поиск по тэгу
    elements_by_tag = driver.find_elements(By.TAG_NAME, value="h1")
    print("With h1 tag:", len(elements_by_tag))
    # Поиск по имени
    elements_by_name = driver.find_elements(By.NAME, value="almond-nut")
    print("With almond-nut name:", len(elements_by_name))
    # Поиск по классу
    elements_by_class = driver.find_elements(By.CLASS_NAME, value="imageContainer")
    print("With imageContainer class:", len(elements_by_class))
    # Поиск по ID
    elements_by_ID = driver.find_elements(By.ID, value="peanut")
    print("With peanut ID:", len(elements_by_ID))
    # Поиск по атрибуту
    elements_by_attribute = driver.find_elements(By.CSS_SELECTOR, value="[data-type='nuts']")
    print("With data-type nuts:", len(elements_by_attribute))
    # Ждем некоторое время (в данном случае, 5 секунд)
    time.sleep(5)
except Exception as e:

```

*Рис.2.2.1 - Код программы element\_search.py*

Далее запускаем код программы, по выполнению которой в консоли выведутся все искомые данные.

```

C:\Users\Keltasar\PycharmProjects\ProjectAT\.venv\Scripts\python.exe C:\Users\Keltasar\PycharmProjects\ProjectAT\element_search.py
With h1 tag: 7
With almond-nut name: 1
With imageContainer class: 6
With peanut ID: 1
With data-type nuts: 6

```

*Рис.2.2.2 - Результат работы программы element\_search.py*



Для проверки корректности выполнения программы можно подсчитать все элементы вручную при помощи DevTools.

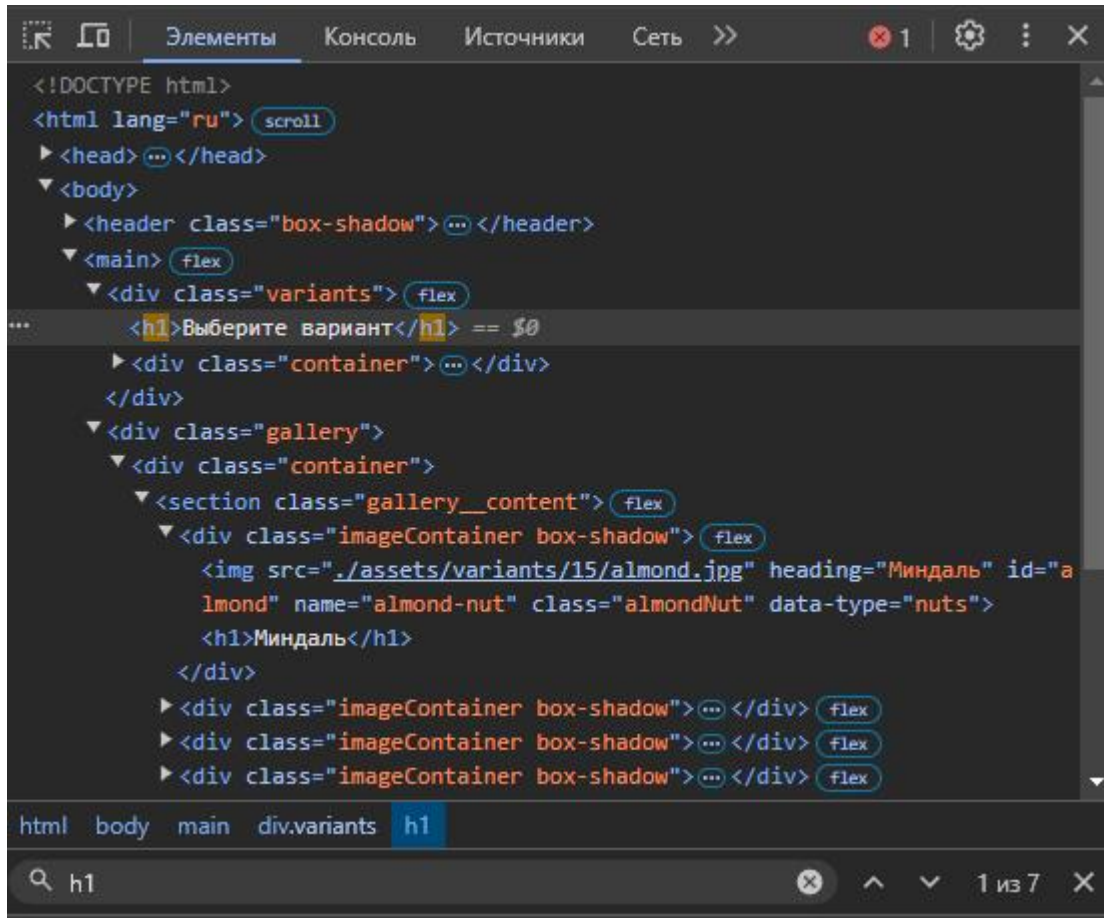


Рис.2.2.3 - Поиск по метке  $h1$

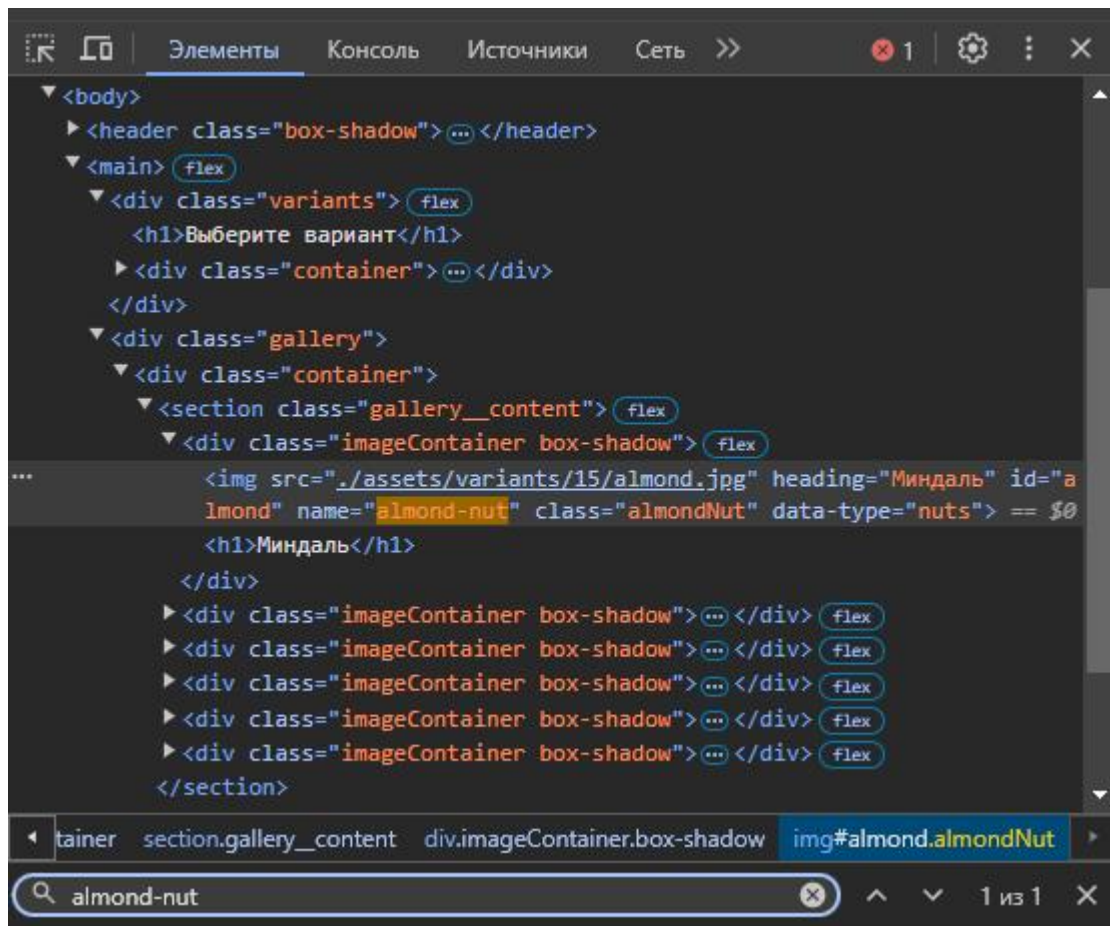


Рис.2.2.4 - Поиск по имени almond-nut

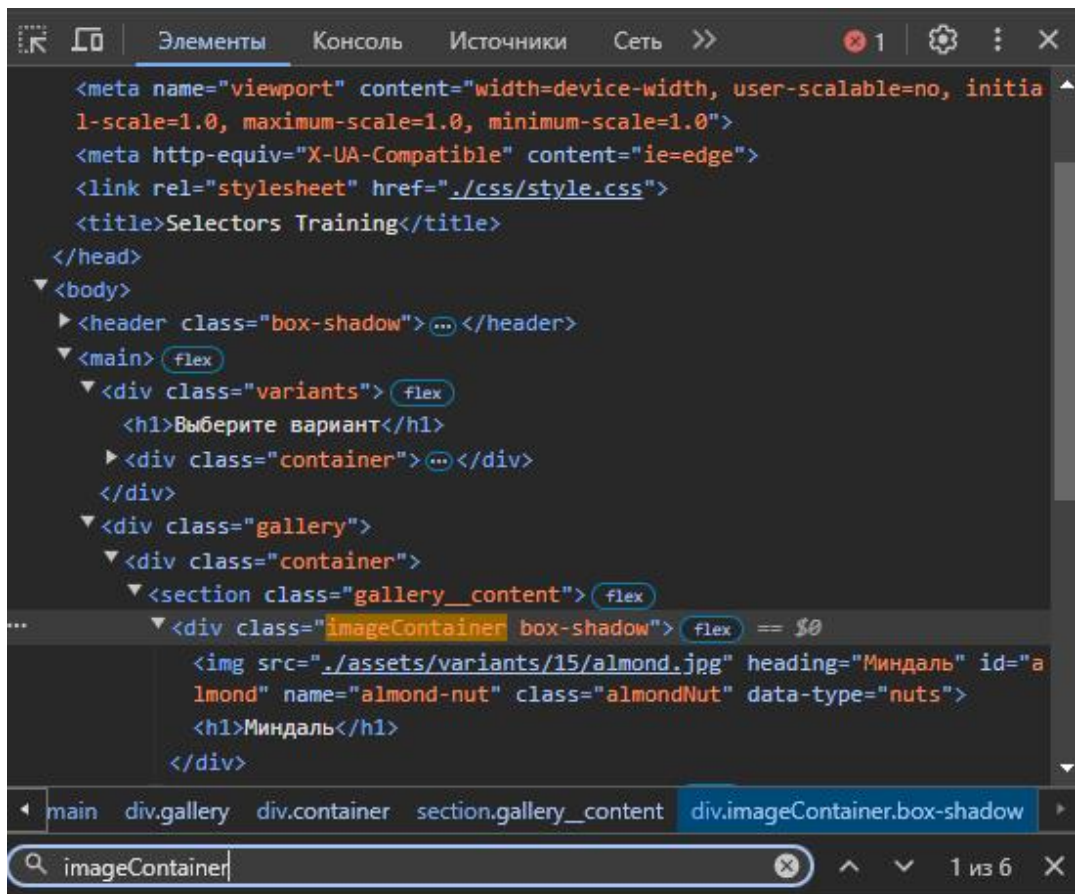


Рис. 2.2.5 - Поиск по классу imageContainer

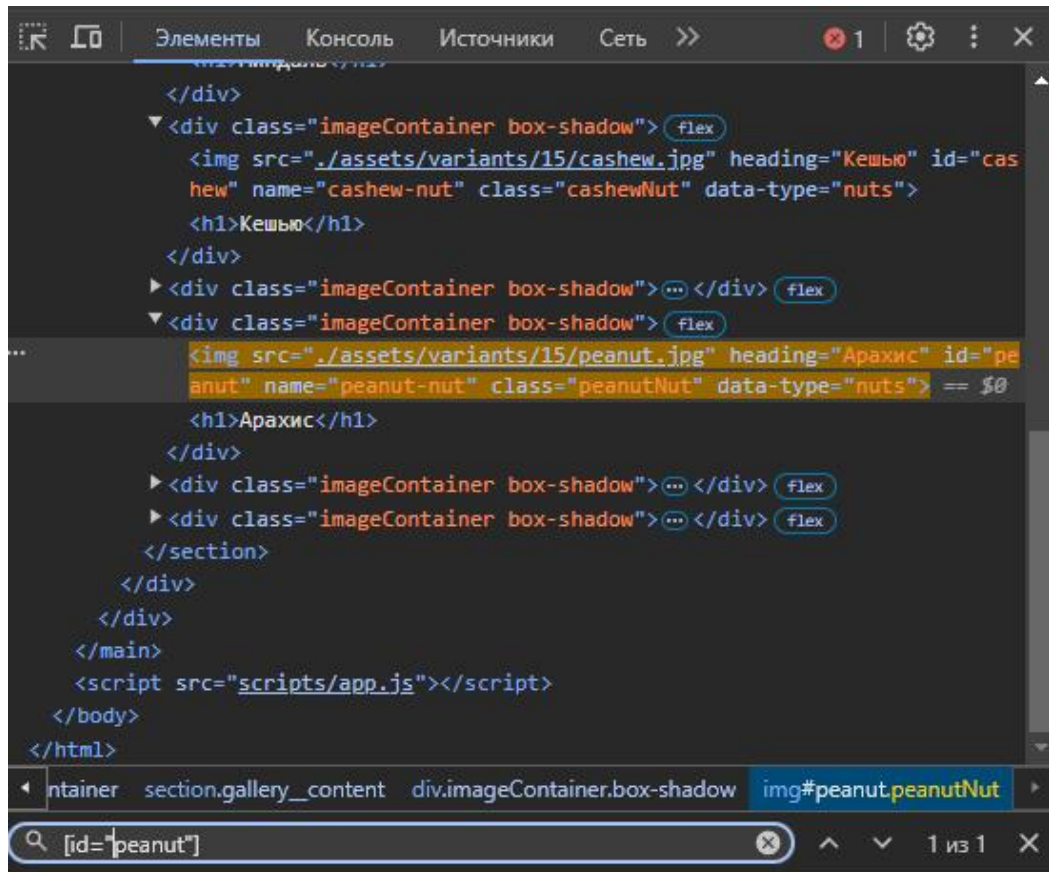


Рис. 2.2.6 - Поиск по ID peanut

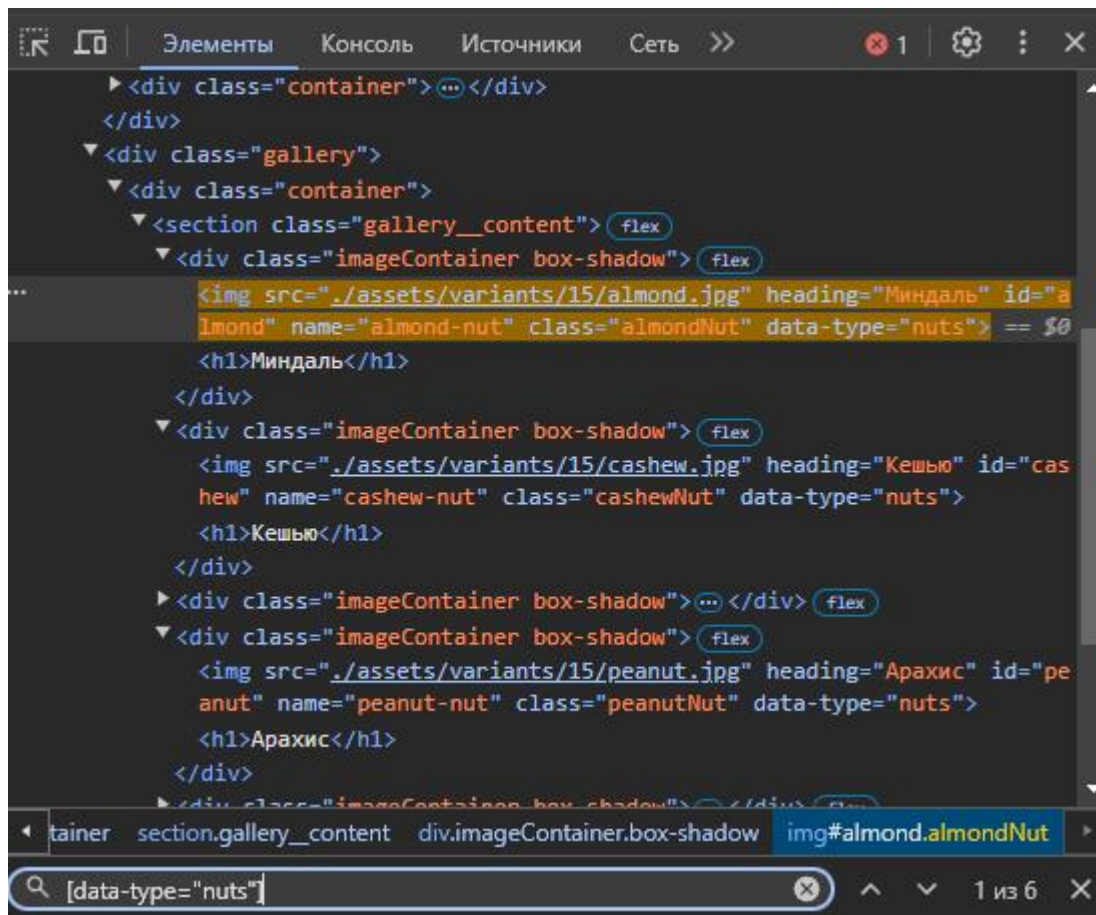


Рис.2.2.7 - Поиск по [data-type="nuts"]

Согласно моим подсчётам программа выполняется верно.

## 2.3. Заполнение полей регистрации

Для выполнения задания с заполнением полей регистрации создаём файл “registration.py”. В нём будет содержаться [ссылка](#) на форму регистрации, которую необходимо заполнить за 5 секунд, и код, заполняющий эту самую форму автоматически.



```
5
6 # Путь к исполняемому файлу chromedriver.exe
7 chrome_driver_path = 'chromedriver-win64/chromedriver.exe'
8 # Создание сервиса Chrome
9 chrome_service = ChromeService(executable_path=chrome_driver_path)
10 # Создание экземпляра браузера
11 driver = webdriver.Chrome(service=chrome_service)
12 # Ссылка на учебный веб-проект
13 URL = "https://qa-course.netlify.app/registration-form-timer"
14 try:
15     # Открытие веб-системы
16     driver.get(URL)
17     # Ввод имени
18     name = driver.find_element(By.NAME, value="firstName")
19     name.send_keys("Artyom")
20     # Ввод фамилии
21     last_name = driver.find_element(By.NAME, value="lastName")
22     last_name.send_keys("Podberyoyskiy")
23     # Ввод города
24     city = driver.find_element(By.NAME, value="city")
25     city.send_keys("Saint-Petersburg")
26     # Отправка данных
27     button = driver.find_element(By.CSS_SELECTOR, value='button[type="submit"]')
28     button.click()
29     # Ждем некоторое время (в данном случае, 5 секунд)
30     time.sleep(5)
31 except Exception as e:
32     print(f"Произошла ошибка: {e}")
33 finally:
```

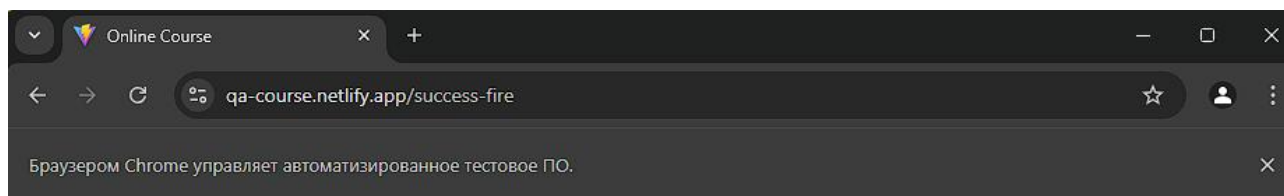
Run registration x

C:\Users\Keltasar\PycharmProjects\ProjectAT\.venv\Scripts\python.exe C:\Users\Keltasar\PycharmProjects\ProjectAT\registration.py

Process finished with exit code 0

*Рис. 2.3.1 - Код программы и отметка об успешном выполнении*

После выполнения программы высвечивается сообщение об успешном прохождении регистрации и в консоли IDE выводится сообщение о выходе с кодом 0, что обозначает, что ошибки при выполнении отсутствуют.



SIMPLE  
SELECTORS

COMPOSITE  
SELECTORS

REGISTRATION

REGISTRATION  
🔥

REGISTRATION  
(5SEC)

# Регистрация пройдена успешно



Рис. 2.3.2 - Сообщение об успешном прохождении регистрации за 5 секунд

## **ВЫВОД**

В ходе практики были успешно выполнены все поставленные задачи: установлены PyCharm Community Edition, Google WebDriver и библиотеки для

Python(selenium и pytest), реализованы и проверены автотесты на открытие/закрытие веб-приложения, автоматический поиск элементов на странице и автоматическое заполнение формы регистрации и её отправка.

Также Selenium Webdriver хорошо показал себя в тестировании веб-приложений методом “черного ящика”. По итогам практики были освоены и применены принципы автоматизации тестирования, которые пригодятся в профессиональной деятельности.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Официальный портал проекта Selenium. URL: <https://www.selenium.dev/> (дата обращения 04.11.2024).
2. Документация по Selenium Webdriver. URL: <https://www.selenium.dev/documentation/webdriver/> (дата обращения 04.11.2024).
3. Документация по Selenium Webdriver. URL: <https://w3c.github.io/webdriver/>
4. Selenium WebDriver. URL: <https://testengineer.ru/selenium-webdriver/> (дата обращения 04.11.2024).
5. Материалы курсы «Автоматизация тестирования с помощью selenium и python». URL: <https://stepik.org/course/575/> (дата обращения 04.11.2024).

## ЛИСТИНГ КОДА 2.1

```
from selenium import webdriver
import time
from selenium.webdriver.chrome.service import Service as ChromeService
# Путь к исполняемому файлу chromedriver.exe
chrome_driver_path = 'chromedriver-win64/chromedriver.exe'
# Создание сервиса Chrome
chrome_service = ChromeService(executable_path=chrome_driver_path)
# Создание экземпляра браузера
driver = webdriver.Chrome(service=chrome_service)
try:
    # Открытие веб-системы
    driver.get('https://2ch.hk/')
    # Ждем некоторое время (в данном случае, 5 секунд)
    time.sleep(5)
except Exception as e:
    print(f'Произошла ошибка: {e}')
finally:
    # Закрытие браузера
    driver.quit()
```

## ЛИСТИНГ КОДА 2.2

```
from selenium import webdriver
import time
from selenium.webdriver.chrome.service import Service as ChromeService
from selenium.webdriver.common.by import By

# Путь к исполняемому файлу chromedriver.exe
chrome_driver_path = 'chromedriver-win64/chromedriver.exe'
# Создание сервиса Chrome
chrome_service = ChromeService(executable_path=chrome_driver_path)
# Создание экземпляра браузера
driver = webdriver.Chrome(service=chrome_service)
# Ссылка на учебный веб-проект и номер варианта
URL = "https://qa-test-selectors.netlify.app/"
variant = 15
try:
    # Открытие веб-системы
    driver.get(URL)
    # Ожидание загрузки всех элементов страницы
    driver.implicitly_wait(10)
    # Нажатие кнопки варианта
    v_button = driver.find_elements(By.TAG_NAME, "button")[variant-1]
    v_button.click()
    # Поиск по тэгу
    elements_by_tag = driver.find_elements(By.TAG_NAME, "h1")
    print("With h1 tag:", len(elements_by_tag))
    # Поиск по имени
    elements_by_name = driver.find_elements(By.NAME, "almond-nut")
    print("With almond-nut name:", len(elements_by_name))
    # Поиск по классу
    elements_by_class = driver.find_elements(By.CLASS_NAME, "imageContainer")
    print("With imageContainer class:", len(elements_by_class))
    # Поиск по ID
    elements_by_ID = driver.find_elements(By.ID, "peanut")
```

```
print("With peanut ID:", len(elements_by_ID))
#Получ по атрибуту
elements_by_attribute = driver.find_elements(By.CSS_SELECTOR, '[data-
type="nuts"]')
print("With data-type nuts:", len(elements_by_attribute))
# Ждем некоторое время (в данном случае, 5 секунд)
time.sleep(5)
except Exception as e:
    print(f"Произошла ошибка: {e}")
finally:
    # Закрытие браузера
    driver.quit()
```

## ЛИСТИНГ КОДА 2.3

```
from selenium import webdriver
import time
from selenium.webdriver.chrome.service import Service as ChromeService
from selenium.webdriver.common.by import By

# Путь к исполняемому файлу chromedriver.exe
chrome_driver_path = 'chromedriver-win64/chromedriver.exe'
# Создание сервиса Chrome
chrome_service = ChromeService(executable_path=chrome_driver_path)
# Создание экземпляра браузера
driver = webdriver.Chrome(service=chrome_service)
# Ссылка на учебный веб-проект
URL = "https://qa-course.netlify.app/registration-form-timer"
try:
    # Открытие веб-системы
    driver.get(URL)
    # Ввод имени
    name = driver.find_element(By.NAME, "firstName")
    name.send_keys("Artyom")
    # Ввод фамилии
    last_name = driver.find_element(By.NAME, "lastName")
    last_name.send_keys("Podbergozskij")
    # Ввод города
    city = driver.find_element(By.NAME, "city")
    city.send_keys("Saint-Petersburg")
    # Отправка данных
    button = driver.find_element(By.CSS_SELECTOR, 'button[type="submit"]')
    button.click()
    # Ждем некоторое время (в данном случае, 5 секунд)
    time.sleep(5)
except Exception as e:
    print(f"Произошла ошибка: {e}")
finally:
```



```
# Заккрытие браузера  
driver.quit()
```