**Знакомство с библиотекой Selenium.**

**Цель:** научится осуществлять поиск элементов веб-страницы с помощью библиотеки selenium

**Задания:**

1. При необходимости, сначала установите библиотеку Selenium, если это еще не сделано:

pip install selenium

Далее, установите пакет webdriver**\_**manager, если он еще не установлен:

pip install webdriver\_manager

2. Необходимо получить секретный код с веб-страницы:

* Откройте заданный сайт <https://parsinger.ru/selenium/1/1.html> с помощью Selenium.
* Используйте метод .find\_elements() для поиска всех доступных полей для ввода на странице.
* В цикле, переберите все найденные поля и заполните их с помощью метода .send\_keys("Текст").
* Найдите кнопку на странице и нажмите на неё.
* Запишите результат в отдельную переменную и выведите её на экран.

3. Найти загадочную ссылку с текстом **16243162441624** на указанной странице

* Откройте заданный сайт <https://parsinger.ru/selenium/2/2.html> с помощью Selenium.
* Используйте метод By.PARTIAL\_LINK\_TEXT или By.LINK\_TEXT для поиска ссылки с текстом **16243162441624**.
* Нажмите на найденную ссылку.
* Скопируйте текст, который появится в теге найденной страницы <p id="result"></p>.
* Запишите результат в отдельную переменную и выведите её на экран.

4. Необходимо получить секретный код с веб-страницы:

* Откройте <https://parsinger.ru/selenium/5/5.html> с помощью Selenium.
* С помощью Selenium и метода click() установите в положение **checked** только те чек-боксы, значение атрибута value которых есть в заданном списке numbers.

numbers = [1, 2, 3, 4, 8, 9, 11, 12, 13, 14, 15, 16, 17, 22, 23, 28, 29, 33, 34, 38, 39, 43, 44, 48, 49, 51, 52, 53, 54, 55, 56, 57, 58, 61, 62, 63, 64, 68, 69, 73, 74, 78, 79, 83, 84, 88, 89, 91, 92, 97, 98, 101, 104, 108, 109, 113, 114, 118, 119, 123, 124, 128, 129, 131, 132, 137, 138, 140, 141, 144, 145, 148, 149, 153, 154, 158, 159, 163, 164, 165, 168, 169, 171, 172, 177, 178, 180, 181, 184, 185, 187, 188, 189, 190, 192, 193, 194, 195, 197, 198, 199, 200, 204, 205, 206, 207, 208, 209, 211, 212, 217, 218, 220, 221, 224, 225, 227, 228, 229, 230, 232, 233, 234, 235, 237, 238, 239, 240, 245, 246, 247, 248, 249, 251, 252, 253, 254, 255, 256, 257, 258, 260, 261, 264, 265, 268, 269, 273, 274, 278, 279, 288, 289, 291, 292, 293, 294, 295, 296, 297, 300, 301, 302, 303, 304, 305, 308, 309, 313, 314, 318, 319, 328, 329, 331, 332, 339, 340, 341, 342, 343, 344, 345, 346, 348, 349, 353, 354, 358, 359, 368, 369, 371, 372, 379, 380, 385, 386, 408, 409, 411, 412, 419, 420, 425, 426, 428, 429, 433, 434, 438, 439, 444, 445, 446, 447, 448, 451, 452, 459, 460, 465, 466, 467, 468, 469, 470, 472, 473, 474, 475, 477, 478, 479, 480, 485, 486, 487, 488, 491, 492, 499, 500, 505, 506, 508, 509, 513, 514, 518, 519]

* Как только все нужные чек-боксы активны, кнопка станет активной. Нажмите на неё.
* После нажатия на кнопку, результат появится в теге <p id="result">Result</p>.
* Запишите результат в отдельную переменную и выведите её на экран.

5. Откройте сайт <https://parsinger.ru/selenium/7/7.html> с помощью Selenium.

* Получите значения всех элементов выпадающего списка.
* Сложите все извлеченные значения.
* Вставьте получившийся результат в поле на сайте и нажмите кнопку.
* Запишите результат в отдельную переменную и выведите её на экран.

**Теоретические сведения**

Класс By — это, по сути, перечисление, которое содержит различные стратегии поиска элементов на веб-странице. Использование By позволяет сделать код более читаемым и поддерживаемым, так как это стандартизирует способы поиска элементов.

* **By.ID** – Поиск элемента по уникальному идентификатору (id). Этот метод очень быстрый и надежный, но требует, чтобы у элемента был атрибут id.

element = driver.find\_element(By.ID, "some\_id")

* **By.CSS\_SELECTOR** – Поиск элемента или элементов, используя селекторы CSS. Это гибкий и мощный метод, который может выразить сложные критерии поиска.

elements = driver.find\_elements(By.CSS\_SELECTOR, ".some\_class")

* **By.XPATH**– Поиск элемента с помощью языка XPath. XPath позволяет создать более сложные запросы, но он менее читаемый и, возможно, будет работать медленнее, чем другие методы.

element = driver.find\_element(By.XPATH, "//div[@attribute='value']")

* **By.NAME** – Поиск элемента по атрибуту name. Этот метод хорошо подходит для форм.

element = driver.find\_element(By.NAME, "username")

* **By.TAG\_NAME** – Поиск элемента по названию HTML-тега. Этот метод полезен, если нужно выбрать, например, все изображения на странице.

images = driver.find\_elements(By.TAG\_NAME, "img")

* **By.CLASS\_NAME** – Поиск элемента или элементов по классу. Этот метод полезен, если у элементов есть общий класс.

buttons = driver.find\_elements(By.CLASS\_NAME, "btn")

* **By.LINK\_TEXT** – Поиск элемента по точному тексту ссылки. Очень удобно, если текст уникален.
* element = driver.find\_element(By.LINK\_TEXT, "Continue")
* **By.PARTIAL\_LINK\_TEXT** – Поиск элемента по частичному тексту ссылки. Удобно, когда точный текст ссылки неизвестен или динамичен.
* element = driver.find\_element(By.PARTIAL\_LINK\_TEXT, "Cont")

Локаторы используются с помощью двух универсальных методов - find\_element(), который возвращает ровно один элемент, найденный первым, и find\_element**s**(), который возвращает список найденных элементов. 

**.find\_element()**

Метод find\_element() используется, когда вам нужно найти один конкретный элемент на странице. Он возвращает первый элемент, который соответствует заданным критериям поиска. Если элемент не найден, Selenium сгенерирует исключение NoSuchElementException.

# Ищем элемент с тегом img

elements = driver.find\_elements(By.TAG\_NAME, 'img')

**.find\_elements()**

Метод find\_element**s**() полезен, когда вы хотите получить список всех элементов, которые соответствуют заданным критериям. В отличие от find\_element(), этот метод вернёт пустой список, если ничего не найдено, вместо того чтобы генерировать исключение.

# Ищем все элементы с классом some\_class

elements = driver.find\_elements(By.CLASS\_NAME, 'some\_class')

Пример кода с поиском элемента и клика по нему.**.find\_element(By.ID, "sale\_button")**

from selenium import webdriver

from selenium.webdriver.common.by import By

import time

browser = webdriver.Chrome()

browser.get('http://parsinger.ru/html/watch/1/1\_1.html')

button = browser.find\_element(By.ID, "sale\_button").click()

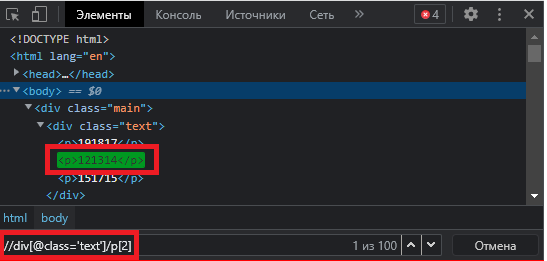
time.sleep(10)

Методы .find\_element() и find\_elements() вы будете использовать всегда при написании скриптов с помощью Selenium. Поэтому их нужно хорошо понимать.

У нас есть [страница](http://parsinger.ru/selenium/3/3.html) с очень простой структурой дерева HTML. На этой странице есть 100 блоков  
<div class="text">, в каждом три тега <p>, которые не имеют ни class, ни id. Допустим мы хотим  собрать каждый первый элемент <p>.

Мы всегда можем решить эту задачу при помощи **XPath.**

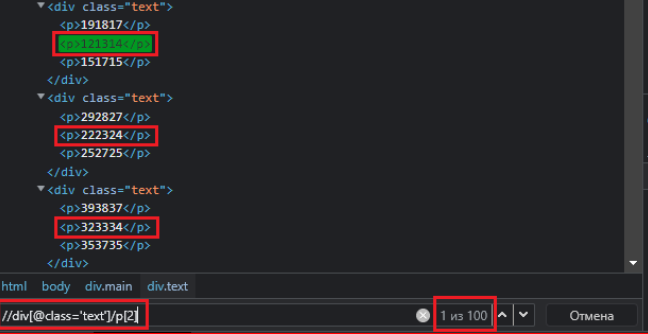
1. .find\_element(By.XPATH, "//div[@class='text']/p[2]") — эта команда найдет второй тег <p> внутри первого тега <div> с классом 'text', который будет обнаружен на странице первым, и вернет его как объект **WebElement**.   
     
   Давай разберем, что означает каждая часть этого XPath выражения **//div[@class='text']/p[2]**.
   * **//**: Символы двойного слэша указывают на то, что нужно искать элемент на всей странице, начиная с корневого элемента.
   * **div[@class='text']**: Эта часть означает, что мы ищем элемент <div>, у которого атрибут class равен 'text'.
   * **/p[2]**: Эта часть указывает, что мы хотим найти второй дочерний элемент <p> внутри ранее найденного <div> с классом 'text'.



.find\_element**s**(By.XPATH, "//div[@class='text']/p[2]") — соответственно, вернёт все найденные элементы <p>, расположенные на вторых позициях, во всех найденных <div class="text">.

В самом XPath выражении **//div[@class='text']/p[2]** происходит следующее:

* **//div[@class='text']**: Этот фрагмент ищет *все* <div> элементы с атрибутом class, значение которого равно 'text'. Здесь двойной слэш // означает, что поиск будет осуществляться по всему дереву DOM, а не только среди дочерних элементов какого-то конкретного элемента.
* **/p[2]**: Этот фрагмент уточняет, что нам нужен именно второй <p> элемент внутри каждого найденного <div> с классом 'text'.



В итоге, .find\_elements(By.XPATH, "**//div[@class='text']/p[2]**") вернет список объектов **WebElement**, каждый из которых будет представлять второй <p> элемент внутри каждого <div> с классом 'text' на странице.  
  
Пример кода:

from selenium import webdriver

from selenium.webdriver.common.by import By

# URL веб-страницы для парсинга

url = 'http://parsinger.ru/selenium/3/3.html'

# Инициализируем драйвер Chrome

with webdriver.Chrome() as browser:

# Открываем веб-страницу по заданному URL

browser.get(url)

# Используем метод .find\_elements() для поиска всех элементов, соответствующих нашему XPath

p\_elements = browser.find\_elements(By.XPATH, "//div[@class='text']/p[2]")

# Проходимся по списку найденных элементов и выводим их текст

for i, p\_element in enumerate(p\_elements):

print(f"Текст второго p тега в {i + 1}-м div с классом 'text': {p\_element.text}")

Это можно сделать ещё проще с использованием относительного пути XPATH.

first\_p = div.find\_element(By.XPATH, './p[1]')

third\_p = div.find\_element(By.XPATH, './p[3]')

from selenium import webdriver

from selenium.webdriver.common.by import By

url = 'http://parsinger.ru/selenium/3/3.html'

# Инициализация драйвера в контексте with, чтобы он закрылся после завершения работы

with webdriver.Chrome() as browser:

# Открываем URL

browser.get(url)

# Ищем все div с классом 'text'

divs = browser.find\_elements(By.CLASS\_NAME, 'text')

# Проходимся по каждому div

for i, div in enumerate(divs):

# Получаем первый и третий теги <p> внутри каждого div

first\_p = div.find\_element(By.XPATH, './p[1]')

third\_p = div.find\_element(By.XPATH, './p[3]')

# Выводим их текст

print(f"Для div #{i+1}, первый p: {first\_p.text}, третий p: {third\_p.text}")

Вывод:

Для div #1, первый p: 191817, третий p: 151715

Для div #2, первый p: 292827, третий p: 252725

Для div #3, первый p: 393837, третий p: 353735

Для div #4, первый p: 494847, третий p: 454745

...

...

...

В этом коде, XPath выражения ./p[1] и ./p[3] относятся к первому и третьему элементу <p> внутри каждого найденного <div> с классом text.

* ./ — указывает на текущий элемент, в контексте которого происходит поиск. В данном случае, текущим элементом является каждый отдельный <div> с классом text.
* p[1] и p[3] — это собственно фильтры, которые указывают, какой именно элемент <p> нам нужен. В XPath, индексация начинается с 1, так что p[1] это первый <p> элемент, а p[3] это третий.

Таким образом, когда вы выполняете div.find\_element(By.XPATH, './p[**1**]'), вы ищете первый тег <p> внутри текущего <div> с классом text. Аналогично, div.find\_element(By.XPATH, './p[**3**]') находит третий тег <p> внутри этого же <div>.

Эти выражения позволяют вам очень гибко и точно указать, какие именно элементы вам нужны, не прибегая к использованию циклов и сложных срезов.

Для заполнения текстового поля используем метод send\_key()

input\_fields = browser.find\_element(By.CLASS\_NAME, 'form')

input\_form = f.send\_keys('Text')