

## Selenium – Basics

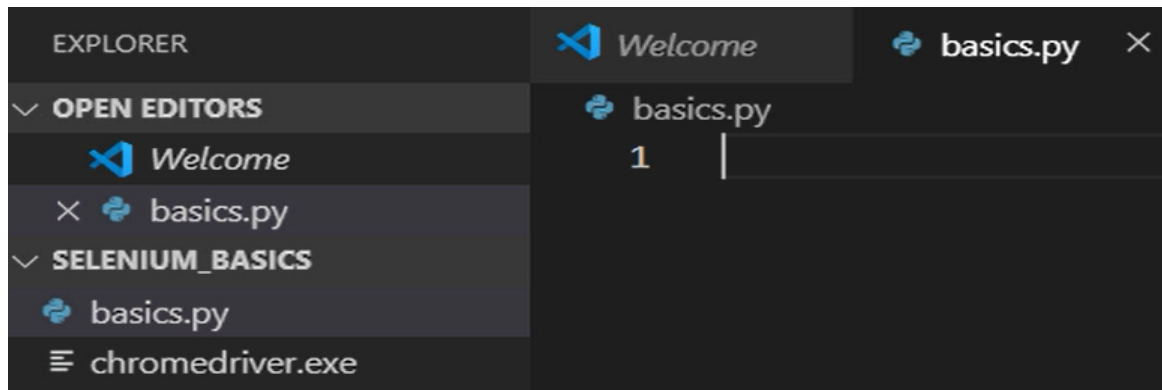
Bir çok insan selenium'u splash'e tercih ediyor çünkü it is more beginner friendly and easy to understand.

Ancak aslında selenium'un amacı web scraping değil, daha çok automation/test tool.

Official documentation: <https://selenium-python.readthedocs.io/>

### Öncelikle installation yapacağız.

- Terminalde projects folder'ı içindeyken `conda install selenium -y`
- Bende bu işe yaramadı `pip install selenium` dedim o yaradı!
- Sıradaki adım, driver'ı yani selenium ile kullanacağımız browser'ı indirmek olacak, aşağıdaki linke gideriz:  
<https://sites.google.com/a/chromium.org/chromedriver/downloads>
- Burada current releases'i görebiliriz ancak hangisini indireceğimiz bizim bilgisayarımızda halihazırda yüklü olan chrome sürümüne bağlı.
- **Chrome Ayarlarından** > **Chrome Hakkında** kısmına gelip versiyonumun 83.0.4103.97 olduğunu görebilirim.
- O halde current releases'dan version 83 olması gerek.
- İlgili versiyonu indirdikten sonra, project klasörüne gidip elle selenium-basics isimindeki klasörümü yarattım.
- İndirdiğim driver içerisindeki driver.exe'yi yarattığım klasörün içine extract ediyorum.
- Daha sonra oluşturulan selenium-basics klasörünü vs code ile açıyorum.
- Daha sonra vs code üzerinden bu dosya içerisinde basics.py isminde yeni bir dosya yaratıyorum.
- Selenium işlemlerini bu dosya üzerinde yapacağım.



Özetle, elle oluşturduğumuz bir proje klasörünün içerisine indiriler driver'ı yani browser exe'sini attık bunun yanında da kodun yazılacağı bir py dosyası yarattık. Artık bu py dosyası içerisinde, driver'a komutlar verebiliriz.

Bu örnek için daha önce splash için kullandığımız Duckduckgo sayfasını kullanacağız burada, form fill etmeyi tıklamayı eleman seçmeyi keyboard key girmeyi vesaire öğreneceğiz.

## Practice on Duckduckgo – Intro

Oluşturulan basics.py dosyasının içi aşağıdaki lejikde dolduruluyor, açıklamaları zaten kod içinde yaptım.

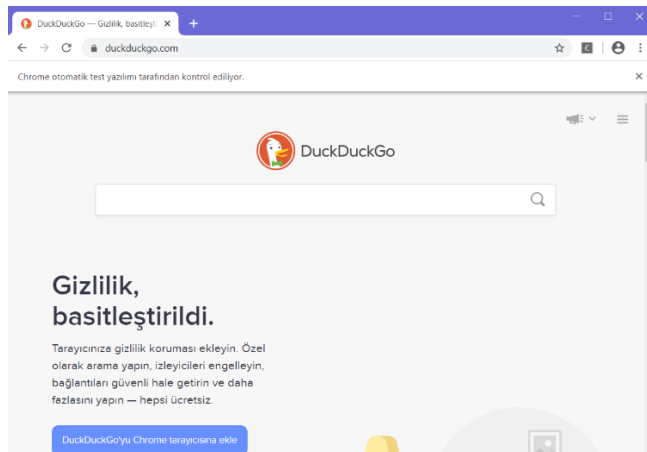
```
1 #selenium içerisinden webdriver paketini import ediyoruz.
2 from selenium import webdriver
3
4 #import edilen paketi kullanarak, indirdiğimiz chrome driver'ını driver
5 #olarak ata diyoruz.
6 driver = webdriver.Chrome(executable_path="./chromedriver.exe")
7
8 #daha sona çok sade bir şekilde, url'yi girerek
9 #driver'a bu url'yi aç diyoruz.
10 #aynen chrome kullanmak gibi ancak kodla kullanıyoruz.
11 driver.get("https://duckduckgo.com")
```

Yukarıdaki executable path belirtirken, chrome driver basics.py ile aynı klasörde olduğu için ./ demek yeterli oldu, olmasaydı path'ı tanımlayacaktık.

Daha sonra bu selenium kodunu çalıştırmak için vscode terminaline:

```
python .\basics.py
```

yazıyorum ve otomatik olarak bir chrome sekmesinin açıldığını ve belirtilen url'yi açtığını görüyorum:



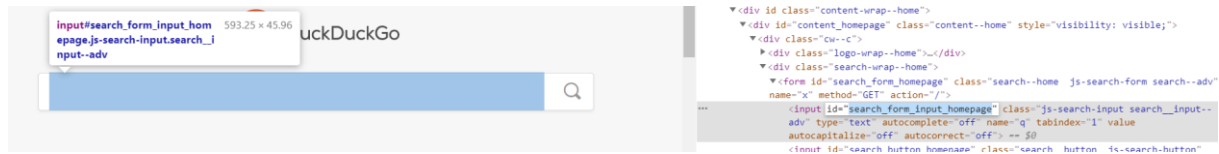
Yukarıdaki kod içinde driver path'ini direkt belirttiğimize dikkat et, buna alternative bir yol da vardır. Bir modül kullanarak, sadece driver'ın adını kullanarak driver path'ini elde edebiliriz ve executable\_path içine de elde edilen bu path'i verebiliriz yani, kod şu hale gelecek:

```
1  #selenium içerisinde webdriver paketini import ediyoruz.
2  from selenium import webdriver
3  #indirilen chrome driver'ının path'ini elde etmek için kullanacağız.
4  from shutil import which
5
6  #chrome driver'ının sadece adını vererek path'i elde ettik.
7  chrome_path = which("chromedriver")
8
9  #indirilen chromedriver'ı driver olarak atadık.
10 driver = webdriver.Chrome(executable_path = chrome_path)
11
12 #driver ile istediğimiz url'i açtık.
13 driver.get("https://duckduckgo.com")
```

Çalıştırırsak aynı sonucu elde ederiz, yani duckduckgo.com'u açan bir chrome browser göreceğiz.

## Practice on Duckduckgo – Filling Forms

Daha önce olduğu gibi öncelikle search input'un elemanını elde ediyorum:



İlgili input elemanının id'sini kopyalıyorum.

Aşağıda selenium ile eleman seçmek için bazı yöntemler yer alıyor:

```
#input elemanını id'sini kullanarak bulalım:
driver.find_element_by_id("search_form_input_homepage")

#alternatif arama yöntemleri:
driver.find_element_by_class_name()#sadece tek bir eleman seçer.
driver.find_elements_by_class_name()#o class'ın tüm elemanlarını seçer.
driver.find_elements_by_tag_name()#tag name'e göre eleman seçer.
driver.find_elements_by_xpath()#xpath ile eleman seçer.
driver.find_elements_by_css_selector()#css selector ile eleman seçer.
#elements ile element farkı tüm methodlarda geçerlidir!
```

Şimdi ilgili elemanı selenium ile seçmek için `driver.find_element_by_id()` methodunu kullanacağım, ardından da `send_keys()` methodu ile seçilen input elemanı içine istediğim text'i yollayacağım:

```
#selenium içerisinde webdriver paketini import ediyoruz.
from selenium import webdriver
#indirilen chrome driver'ının path'ini elde etmek için kullanacağız.
from shutil import which

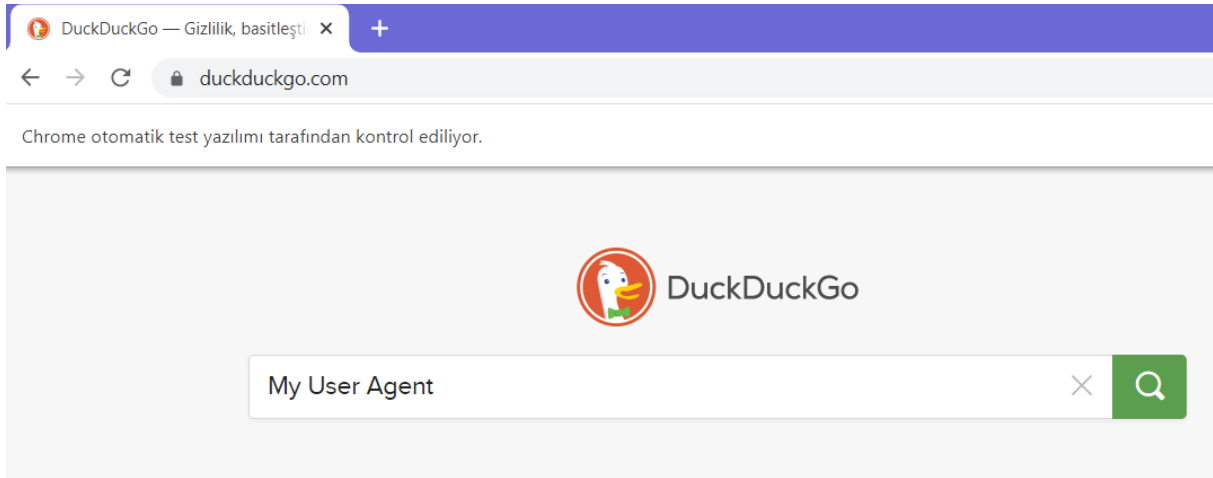
#chrome driver'ının sadece adını vererek path'i elde ettik.
chrome_path = which("chromedriver")

#indirilen chromedriver'ı driver olarak atadık.
driver = webdriver.Chrome(executable_path = chrome_path)

#driver ile istediğimiz url'i açtık.
driver.get("https://duckduckgo.com")

#input elemanını id'sini kullanarak bulalım:
search_input = driver.find_element_by_id("search_form_input_homepage")
search_input.send_keys("My User Agent")
```

Ardından `python .\basics.py` ile hazırlanan selenium kodunu çalıştırırsam, görüyorum ki driver'ım istediğimi yapıyor:



## Practice on Duckduckgo – Press Enter

Formu doldurduktan sonra enter'a basarak aramamızı yapabiliriz, bunun için Keys module'ünü import ediyorum ve search input'a ENTER keyini gönderiyorum, yani bu input seçiliyken ENTER'a basmış oluyorum.

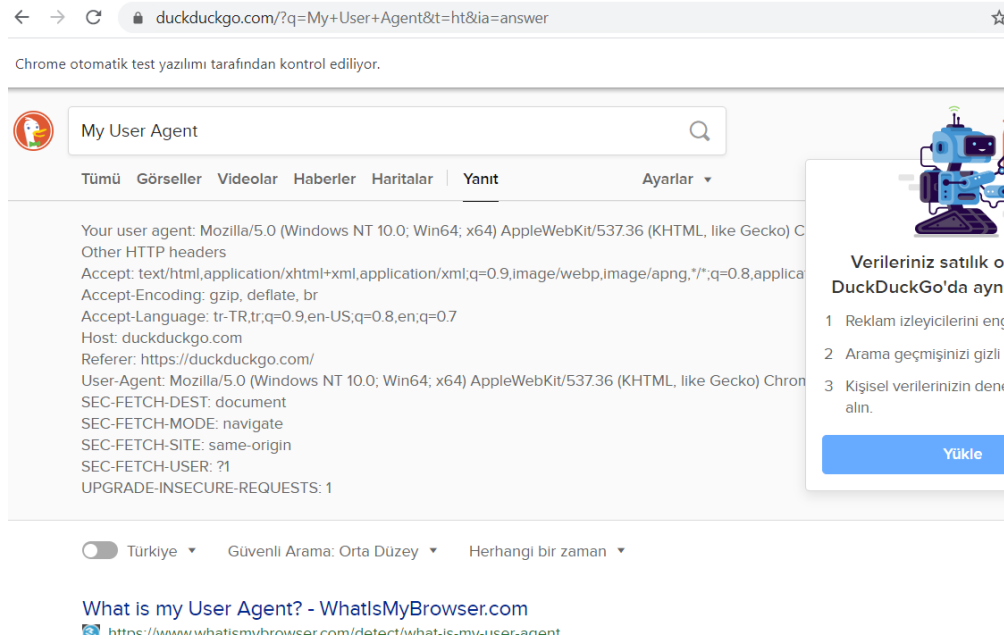
```
2  from selenium import webdriver
3  #indirilen chrome driver'ının path'ini elde etmek için kullanacağız.
4  from shutil import which
5  #Enter key'ini gönderebilmek için bunu import ediyorum:
6  from selenium.webdriver.common.keys import Keys
7
8  #chrome driver'ının sadece adını vererek path'i elde ettik.
9  chrome_path = which("chromedriver")
10
11 #indirilen chromedriver'ı driver olarak atadık.
12 driver = webdriver.Chrome(executable_path = chrome_path)
13
14 #driver ile istediğimiz url'i açtık.
15 driver.get("https://duckduckgo.com")
16
17 #input elemanını id'sini kullanarak bulalım:
18 search_input = driver.find_element_by_id("search_form_input_homepage")
19 search_input.send_keys("My User Agent")
20
21 #butona basmak yerine, enter tuşuna basalım:
22 search_input.send_keys(Keys.ENTER)
```

## Practice on Duckduckgo – Clicking

Şimdi ise ENTER'a basarak arama yapmak yerine ekrandaki search button'ına tıklamak istiyorum:

```
2 from selenium import webdriver
3 #indirilen chrome driver'ının path'ini elde etmek için kullanacağız.
4 from shutil import which
5
6 #chrome driver'ının sadece adını vererek path'i elde ettik.
7 chrome_path = which("chromedriver")
8
9 #indirilen chromedriver'ı driver olarak atadık.
10 driver = webdriver.Chrome(executable_path = chrome_path)
11
12 #driver ile istediğimiz url'i açtık.
13 driver.get("https://duckduckgo.com")
14
15 #input elemanını id'sini kullanarak bulalım:
16 search_input = driver.find_element_by_id("search_form_input_homepage")
17 search_input.send_keys("My User Agent")
18
19 #şimdi de search button'ını seçip elemana tıklayalım:
20 search_btn = driver.find_element_by_id("search_button_homepage")
21 search_btn.click()
22
```

Basitçe, buton elemanını seçtim ve click() methodunu uyguladım. Sonuçta kodu çalıştırınca görüyoruz ki, işlemler gerçekleştiriliyor:



## Practice on Duckduckgo – Best Practices

İlk konu şu ne zaman bir driver instantiate etsek, işimiz bitince `driver.close()` ile kapatmayı unutmamalıyız.

Gördüğümüz gibi yukarıdaki örnekte, kod uygulanırken browser açılıyor ve biz adımları takip edebiliyoruz, gerçekte biz bir uygulamayı müşteriye teslim ederken bunun gerçekleşmesini istemeyiz. Bu yüzden **HEADLESS BROWSER** kullanırız.

Bunu sağlamak için kodda bazı değişiklikler yapmam gerekecek.

```
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.chrome.options import Options
from shutil import which

chrome_options = Options()
chrome_options.add_argument("--headless")

chrome_path = which("chromedriver")

driver = webdriver.Chrome(executable_path=chrome_path, options=chrome_options)
driver.get("https://duckduckgo.com")

search_input = driver.find_element_by_id("search_form_input_homepage")
search_input.send_keys("My User Agent")

search_input.send_keys(Keys.ENTER)

print(driver.page_source)

driver.close()
```

- Öncelikle Options paketi import edildi ve headless option'ı set edildi.
- Bu işlem yapıldıktan sonra kodu çalıştırsak, göreceğiz ki eskiden açılan browser penceresi artık açılmıyor işlem under the hood gerçekleştiriliyor.
- Sondaki `driver.page_source` ile ise en son elde edilen sayfanın html markup'ını elde ettik ve konsolda yazdırdık.
- Son olarak işimiz bitince driver'ı kapattık.

## Selenium – ElementNotInteractable Exception

Bende problem olmadı ama, sanıyorum bazı versiyonlarda input elemanını aşağıdaki gibi seçmek, yine aşağıda görülen `elementnotinteractableexception`'a neden oluyor.

Yani bu exception şu demek: ilgili eleman html markup'da mevcut ama bu eleman ile interact edilemiyor, yani bu inputa key gönderemeyiz text gönderemeyiz vesaire.

```
3
4 search_input = driver.find_element_by_id("search_form_input_homepage")
5 search_input.send_keys("My User Agent")
6
7 # search_btn = driver.find_element_by_id("search_button_homepage")
8 # search_btn.click()
9
10 search_input.send_keys(Keys.ENTER)
11
12 print(driver.page_source)
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

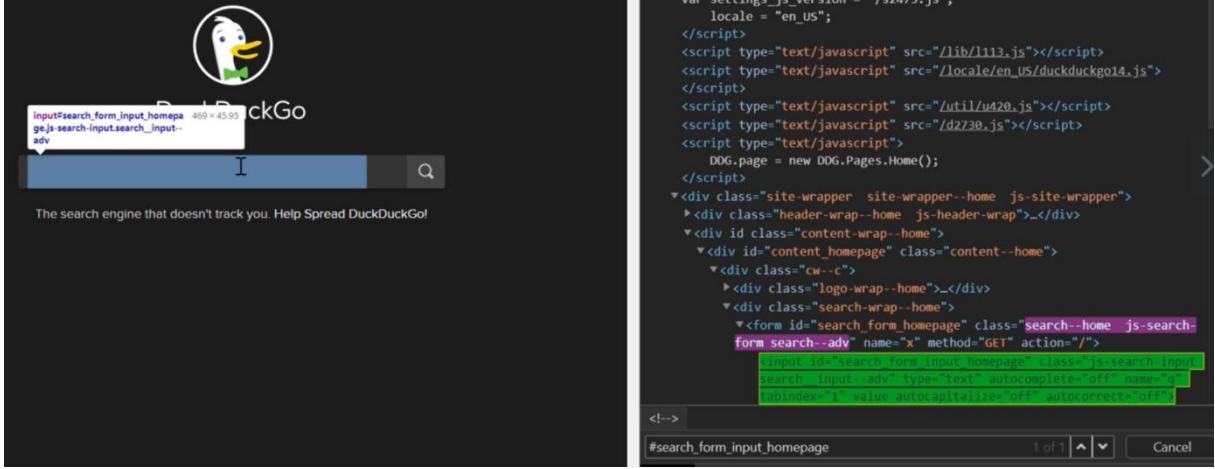
```
C:\Users\Ahmed\projects\selenium_basics> python .\basics.py

vTools listening on ws://127.0.0.1:50467/devtools/browser/157c1189-cf55-4b94-a220-d297bb759cc2
Traceback (most recent call last):
  File ".\basics.py", line 15, in <module>
    search_input.send_keys("My User Agent")
  File "C:\Users\Ahmed\Anaconda3\envs\virtual_workspace\lib\site-packages\selenium\webdriver\remote\webelement.py", line 100, in send_keys
    'value': keys_to_typing(value)))
  File "C:\Users\Ahmed\Anaconda3\envs\virtual_workspace\lib\site-packages\selenium\webdriver\remote\webdriver.py", line 261, in execute
    return self._parent.execute(command, params)
  File "C:\Users\Ahmed\Anaconda3\envs\virtual_workspace\lib\site-packages\selenium\webdriver\remote\webdriver.py", line 271, in _parent.execute
    self.error_handler.check_response(response)
  File "C:\Users\Ahmed\Anaconda3\envs\virtual_workspace\lib\site-packages\selenium\webdriver\remote\errorhandler.py", line 24, in check_response
    raise exception_class(message, screen, stacktrace)
selenium.common.exceptions.ElementNotInteractableException: Message: element not interactable
(Session info: chrome=79.0.3943.130)
```

Yani olay şu: bizim kendi browser'ımızda ilgili id'li eleman seçilebiliyor ve interactable ancak, driver'ın açtığı browser ekranında aynı elemanı seçersek görüyoruz ki browserda eleman seçilmiyor.

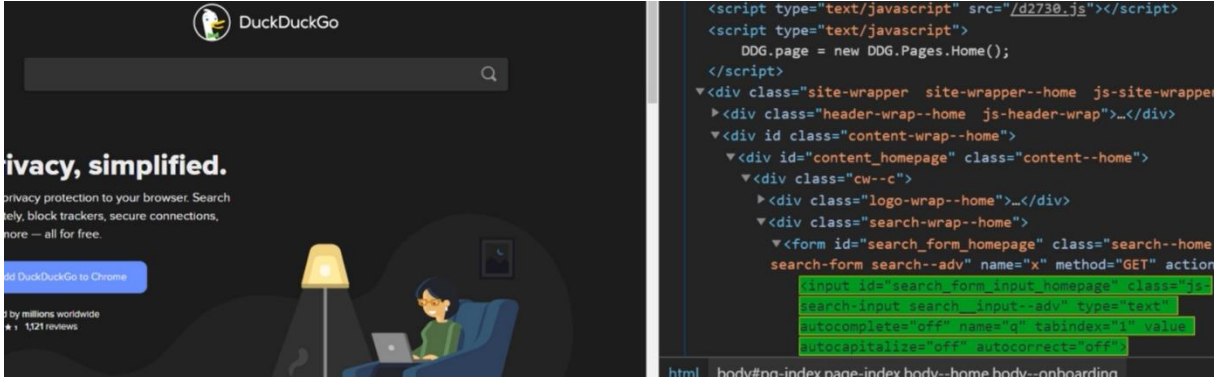


Bunu anlamak için, önce kendi browserımızdan duckduckgo'ya gidip developer'dan ilgili id'li eleman seçiliyor.



Göründüğü gibi eleman browserda da seçiliyor.

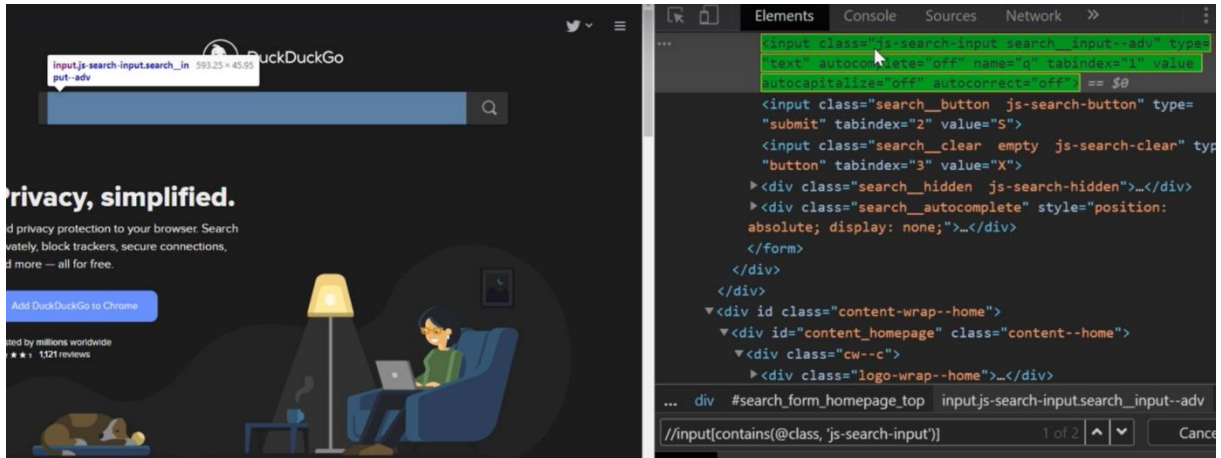
Şimdi açılan driver üzerinden developer ile aynı elemanı seçmeye çalışalım:



Göründüğü gibi html markupda eleman yer alsa da driver üzerinde eleman seçilmiyor yani elemanla interact edemiyoruz!

**BU YÜZDEN ELEMAN SEÇERKEN, TESTİNİ KENDİ BROWSER'INDA DEĞİL DE DRIVER ÜZERİNDE YAPMAMIZ GEREK!**

Bu problemi nasıl çözeceğiz? Eleman seçimini driver üzerinden yapacağız, bu örnek için buna baktığımızda elemanı id ile seçemiyoruz bu yüzden bir xpath expression kullanarak aynı elemanı seçeceğiz:



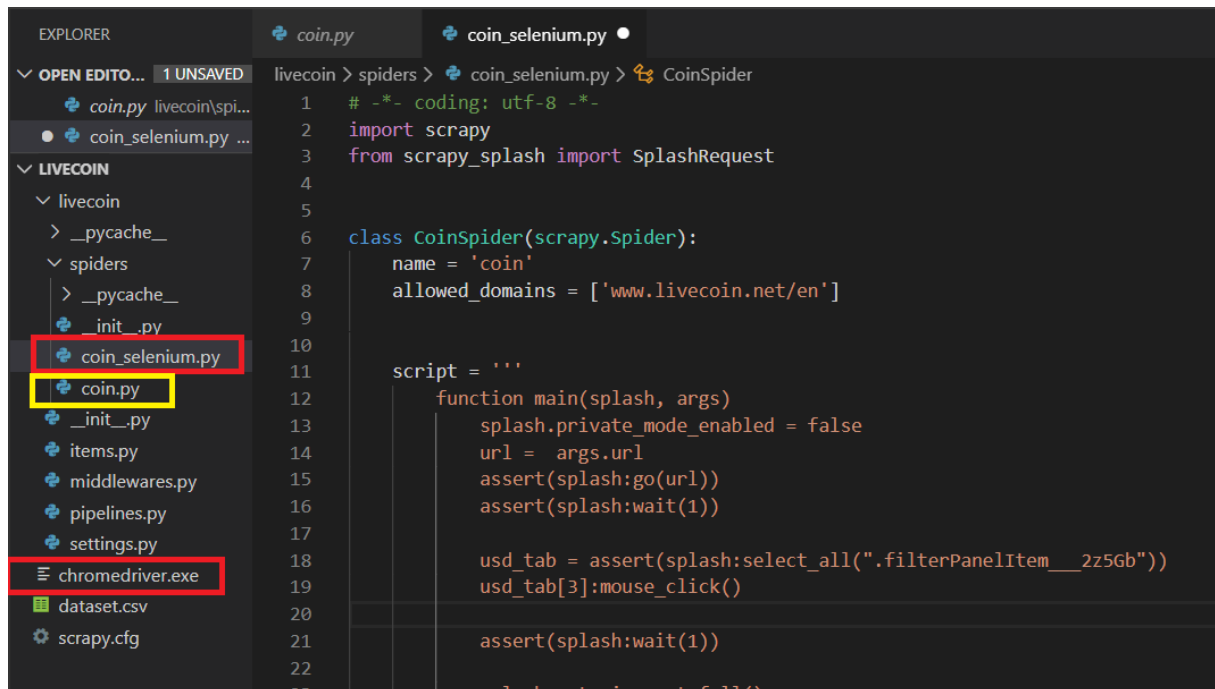
Sonuçta elemanı xpath ile nasıl seçeceğimiz anladık ve, artık bu seçimi selenium kodu içerisinde aşağıdaki gibi gerçekleştirebiliriz.

```
search_input = driver.find_element_by_xpath("(//input[contains(@class, 'js-search-input')])[1]")
search_input.send_keys("My User Agent")
```

## Selenium – Selenium with Scrapy

Şimdi scrapy-selenium kullanarak daha önce scrapy-splash ile scrape ettiğimiz <https://www.livecoin.net/en> sayfasını scrape edeceğiz hatta aynı proje içerisinde yeni bir spider yaratacağız.

- Öncelikle chromedriver.exe'yi bu proje klasörü içine yapıştırıyorum, daha sonra lazım olacak.
- Sonra scrapy-splash ile hazırlanan coin spider'ının içeriğini coin.py'den kopyalıyorum ve coin\_selenium.py adında yeni oluşturduğumuz spider'a yapıştırıyorum. Yani genspider ile yeni spider oluşturmak yerine elle işlem yapıyorum.



```
1  # -*- coding: utf-8 -*-
2  import scrapy
3  from scrapy_splash import SplashRequest
4
5
6  class CoinSpider(scrapy.Spider):
7      name = 'coin'
8      allowed_domains = ['www.livecoin.net/en']
9
10
11     script = '''
12         function main(splash, args)
13             splash.private_mode_enabled = false
14             url = args.url
15             assert(splash:go(url))
16             assert(splash:wait(1))
17
18             usd_tab = assert(splash:select_all(".filterPanelItem__2z5Gb"))
19             usd_tab[3]:mouse_click()
20
21
22             assert(splash:wait(1))
23
24             splash:set_viewport_full()
```

Şimdi coin\_selenium içeriğini değiştireceğim ve scrapy-splash yerine aynı uygulamayı scrapy-selenium ile yapacağım.

## Scrapy-Splash kodunu hatırlayalım:

```
# -*- coding: utf-8 -*-
import scrapy
from scrapy_splash import SplashRequest
```

```
1 # -*- coding: utf-8 -*-
2 import scrapy
3
4
5 class CoinSpider(scrapy.Spider):
6     name = 'coin'
7     allowed_domains = ['www.livecoin.net/en']
8     start_urls = ['http://www.livecoin.net/en/']
9
10    script = '''
11        function main(splash, args)
12            splash.private_mode_enabled = false
13            url = args.url
14            assert(splash:go(url))
15            assert(splash:wait(1))
16            rur_tab = assert(splash:select_all(".filterPanelItem__2z5Gb"))
17            rur_tab[5]:mouse_click()
18            assert(splash:wait(1))
19            splash:set_viewport_full()
20            return splash:html()
21        end
22
23    '''
24
```







Öncelikle initial request için start\_request'i override edeceğiz, çünkü splash ile initial request gönderiliyor bu yüzden start\_urls silinebilir.

```
def start_requests(self):
    yield SplashRequest(url="https://www.livecoin.net/en", callback=self.parse, endpoint="execute", args={
        'lua_source': self.script
    })

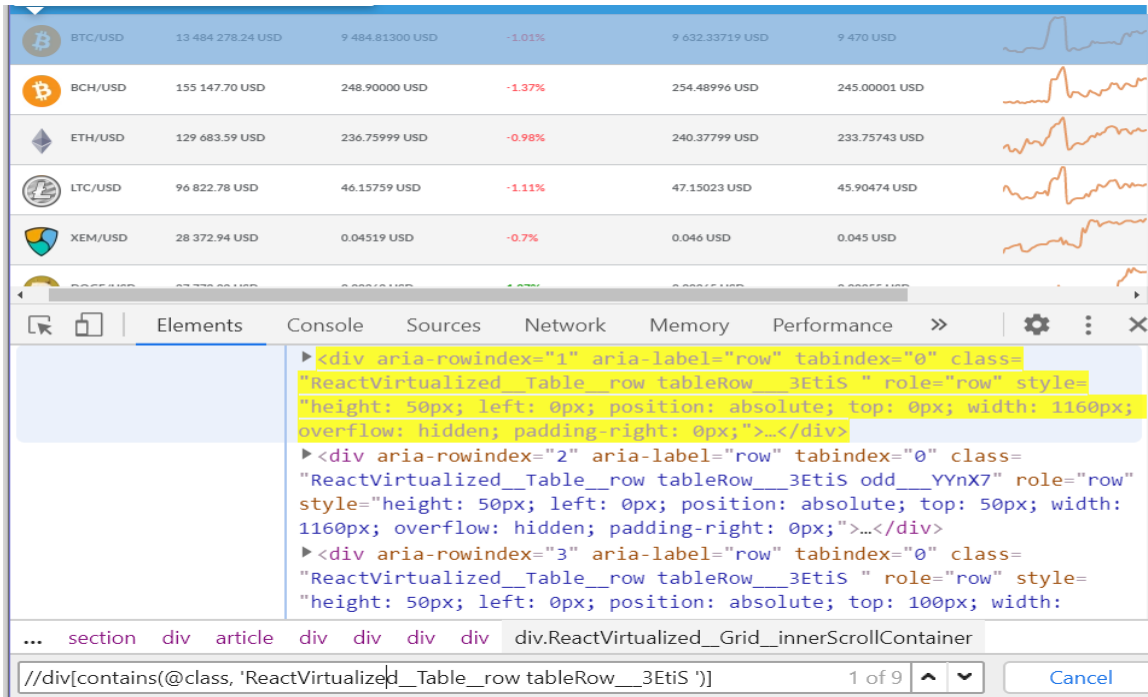
def parse(self, response):
    for currency in response.xpath("//div[contains(@class, 'ReactVirtualized_Table_row tableRow__3Etis ')]"):
        yield {
            'currency pair': currency.xpath("//div[1]/div/text()").get(),
            'volume(24h)': currency.xpath("//div[2]/span/text()").get()
        }
```

Yukarıdaki gibi start request override edildi, bunu şöyle düşünüyorum buradaki SplashRequest ile yukarıda belirtilen script çalıştırılmış oldu, ve kod içindeki url'ye request gönderildi response ise sonuç sayfasının html markup'ı olarak parse methoduna yollandı.

Burada sonuç sayfası dediğimiz aşağıdaki sayfa, yani splash ile URLye gidildi USD sekmesine basıldı, daha sonra elde edilen sayfanın html markup'ı parse methoduna yollandı.

	BTC	USDT	USD	ETH	LTC	Other	
							Search
	Pair ▲ ▼	Volume (24h) ▲ ▼	Last price ▲ ▼	Change (24h) ▲ ▼	High (24h) ▲ ▼	Low (24h) ▲ ▼	Price Graph (7d)
	 BTC/USD	13 589 839.73 USD	9 489.33300 USD	-1.17%	9 632.33719 USD	9 470 USD	
	 BCH/USD	155 147.70 USD	248.90000 USD	-1.37%	254.48996 USD	245.00001 USD	
	 ETH/USD	129 734.96 USD	236.75999 USD	-0.5%	240.37799 USD	233.75743 USD	

Parse methodu içerisindeki response'a xpath selector uygulayarak aşağıdaki gibi her bir satırı yine bir selector objesi olarak elde ettik ve bu selector objelerine de yine xpath uygulayarak currency pair ile volume değerleri elde edildi.



The screenshot shows a web browser displaying a table of cryptocurrency prices. The table has columns for Pair, Volume (24h), Last price, Change (24h), High (24h), Low (24h), and Price Graph (7d). The first row is highlighted in blue and contains the following data: BTC/USD, 13 484 278.24 USD, 9 484.81300 USD, -1.01%, 9 632.33719 USD, 9 470 USD, and a price graph. The second row is highlighted in orange and contains: BCH/USD, 155 147.70 USD, 248.90000 USD, -1.37%, 254.48996 USD, 245.00001 USD, and a price graph. The third row is highlighted in grey and contains: ETH/USD, 129 683.59 USD, 236.75999 USD, -0.98%, 240.37799 USD, 233.75743 USD, and a price graph. The fourth row is highlighted in grey and contains: LTC/USD, 96 822.78 USD, 46.15759 USD, -1.11%, 47.15023 USD, 45.90474 USD, and a price graph. The fifth row is highlighted in grey and contains: XEM/USD, 28 372.94 USD, 0.04519 USD, -0.7%, 0.046 USD, 0.045 USD, and a price graph. The Chrome DevTools Elements panel is open, showing the DOM tree. The selected element is a 

with the following attributes: `aria-rowindex="1" aria-label="row" tabindex="0" class="ReactVirtualized__Table__row tableRow__3Etis" role="row" style="height: 50px; left: 0px; position: absolute; top: 0px; width: 1160px; overflow: hidden; padding-right: 0px;". The XPath selector shown in the bottom bar is //div[contains(@class, 'ReactVirtualized__Table__row tableRow__3Etis ')].`

## Aynı uygulamayı Selenium ile yapalım:

```
# -*- coding: utf-8 -*-
import scrapy

#Selector'ı string olan selenium response'unu selector'a çevirmek için kullanacağız.
from scrapy.selector import Selector
from selenium import webdriver
from selenium.webdriver.chrome.options import Options
from shutil import which

class CoinSpiderSelenium(scrapy.Spider):
    name = 'coin_selenium'
    allowed_domains = ['www.livecoin.net/en']

    #start_request class'ı override edilmediği için default olarak aşağıdaki adrese request gönderilir.
    #response da default olarak parse methodunda yakalanır.
    start_urls = [
        'https://www.livecoin.net/en'
    ]

    #selenium işlemleri consturctor içerisinde yapılacaktır:
    def __init__(self):
        #önce headless driver için ayar yapalım:
        chrome_options = Options()
        chrome_options.add_argument("--headless")

        #driver'ın path'ini ve options'ı vererek driver objemizi elde edelim:
        chrome_path = which("chromedriver")
        driver = webdriver.Chrome(executable_path = chrome_path, options=chrome_options)

        #sayfadaki tüm elemanları tek seferde görebilmek için resolution'ı artıralım.
        driver.set_window_size(1920, 1080)
        #ana url'e giriş yapalım:
        driver.get("https://www.livecoin.net/en")

        #sekmelerden USD sekmesini seçelim ve tıklayalım, ilgili class'a sahip olan 3. eleman USD tab'i oluyor.
        usd_tab = driver.find_elements_by_class_name("filterPanelItem__2z5Gb")
        usd_tab[2].click()

        #sonuçta karşımıza çıkan sayfanın html markup'ını string olarak self.html adında kaydedelim.
        self.html = driver.page_source

        #iş biten driver'ı kapatalım.
        driver.close()

    def parse(self, response):
        #catch edilen response bize lazım değil, bize asıl lazım olan response
        #selenium içinde elde edilen self.html içine kaydedilen html markup'ı
        #ancak bu markup şuan string formatında bunu selector object'e çeviriyoruz:
        resp = Selector(text = self.html)

        #daha sonra bu sayfa üzerinde önceden olduğu gibi tüm satır elemanlarını seçiyoruz
        #ve for loop ile tek tek satırları dolanıp gerekli elemanları yield ediyoruz:
        for currency in resp.xpath("//div[contains(@class, 'ReactVirtualized__Table__row tableRow__3EtiS ')]"):
            yield {
                "currency pair": currency.xpath("./div[1]/div/text()").get(),
                "volume(24h)": currency.xpath("./div[2]/span/text()").get()
            }
```

Yukarıdaki kod üzerinde zaten açıklamalar var ama yine de ekstra bazı açıklamalar yapayım

- Öncelikle tüm selenium işlemlerini constructor içinde yaptık.
  - Splash'deki gibi Selenium ile request göndermedik, selenium kodu zaten direkt bağımsız olarak gitti sayfayı açtı, html markup'ı elde etti.
  - Elde edilen bu html markup'ı parse methoduna pass etmek için self.html attribute'u oluşturuldu.
  - Ancak spider içinden request yollanmazsa parse methodu çalışmayacağı için, star\_urls'i doldurduk ve constructor'dan sonra ilgili url'e scrapy kendi request gönderdi, biz bu request'in response'u ile ilgilenmedik, onun yerine self.html'i string formatından selector formatına çevirdik.
  - Yani bu request'in tek amacı parse methodunu aktive etmektir, kalan işlemler eskisi ile aynı.
- 
- url'ye gitmeden önce browser resolution'ı ayarladık böylece sayfadaki tüm elemanları elde etmek istedik, bunu yapmazsak, diğer sayfa butonu görünebilir ve tek sefer tüm elemanları elde edemeyebiliriz.
- 
- Unutma splash ile doğrudan browser'ı işe başlatamamıştık, start\_request methodunu override ettik ve splash request içerisinde splash kodunu göstererek, spider'ın ilk request'ini splash ile göndermesini sağlamıştık ve response'u parse ile yakalayacağımızı belirtmiştik burada bunlara gerek kalmadı, selenium kodu direkt çalıştı ancak sonucunu parse'a geçirmek için scrapy'nin kendi request'ini kullandık.

## Selenium – Selenium Middleware

Yukarıdaki örnekle yapılan şey şuydu, selenium ile bir url'ye git, açılan sayfada bir butona tıkla, alınan html markup'ı response olarak elde et, daha sonra parse methodunda response html markup'ından elemanları elde et ve yield et.

Peki ya pagination gerekli olursa? Tüm pagination işlemlerini constructor içinde yapmak problem olabileceği için, yani parse methodunun içinde elde edilen bir linke request göndermek isteyebileceğimiz için, seleniumRequest() methoduna ihtiyacımız var.

Splash için bu methodu SplashRequest() idi ve start\_requests'i override ederek zaten initial request'i bu method ile splash tarafından göndermiştik. Şimdi bunun benzerini Selenium için elde edeceğiz ve kullanımını göreceğiz.

### Peki bunun için ne yapacağız?

Selenium ile request yollayabilmek için scrapy-selenium ismindeki downloader middleware'i kullanacağız. Böylece request göndermek için scrapy.Request()'i kullanmak yerine selenium request'i kullanacağız.

- Öncelikle virt\_env içinde terminalde: pip install scrapy-selenium komutunu çalıştırarak paketi indiriyoruz.

- Daha sonra project klasörü içerisinde yeni bir project ve spider yaratalım:

```
scrapy startproject slickdeals
cd slickdeals
scrapy genspider example example.com
```

-Daha sonra vscode ile ilgili projeyi açabiliriz.

-Ardından chromedriver.exe'yi proje klasörüne taşıyoruz.

-Hazırlıkları tamamlamak için de settings.py içerisine en sona aşağıdaki satırları ekliyoruz:

```
#SELENIUM
from shutil import which

SELENIUM_DRIVER_NAME = 'chrome'
SELENIUM_DRIVER_EXECUTABLE_PATH = which('chromedriver')
SELENIUM_DRIVER_ARGUMENTS=['-headless'] # '--headless' if using chrome instead of firefox
```



-Yine settings.py içerisine DOWNLOADER\_MIDDLEWARES = {...} kısmını uncomment edip, aşağıdaki kodu yapıştırıyorum:

```
# Enable or disable downloader middlewares
# See https://doc.scrapy.org/en/latest/topics/downloader-middleware.html
DOWNLOADER_MIDDLEWARES = {
    'scrapy_selenium.SeleniumMiddleware': 800
}
```

Böylelikle selenium request'i scrapy ile kullanabilmek için hazırlıklar tamam.

### Hazırlıklar tamam, spider'a başlayalım – duckduckgo :

```
import scrapy
from scrapy_selenium import SeleniumRequest

class ExampleSpider(scrapy.Spider):
    name = 'example'

    def start_requests(self):
        yield SeleniumRequest(
            url='https://duckduckgo.com',
            wait_time=3,
            screenshot=True,
            callback=self.parse
        )

    def parse(self, response):
        img = response.meta['screenshot']

        with open('screenshot.png', 'wb') as f:
            f.write(img)
```

- Initial request için seleniumRequest'i kullanacağız bu yüzden **start\_urls** listesini ve **allowed\_domains** listesini siliyorum.
- Daha sonra **start\_request()** methodunu override ediyorum, initial request olarak **SeleniumRequest()** gönderiyorum. SeleniumRequest scrapy.Request() veya SplashRequest() ile aşağı yukarı aynı argümanları alır, sadece birkaç argüman fazlası vardır.
  - Örneğin wait\_time argümanı ile response'u render etmeden önce kaç saniye bekleyeceğimizi belirtebiliriz, splash:wait() ile aynı.
  - Response'un screenshot'ını yine bir argümanı set ederek alabiliriz.

- Script ile javascript code execute edebiliriz.
  - Headers ile request headers'ı manipüle edebiliriz.
  - Callback ile de response'un nerede catch edileceğini belirtiyoruz.
- 
- Sonuçta initial request SeleniumRequest ile gönderiyoruz, ilgili url'i belirtiyoruz, response'u render etmeden önce 3 saniye bekle diyoruz ve screenshot=True diyerek response'un screenshot'ını çekmiş oluyoruz. Elde edilen url response'unu da parse methodunda yakalayacağımızı belirtiyoruz.
  - Initial request içerisinde çekilen screenshot'a ulaşmak ve bunu yazdırmak istersek parse içerisindeki kodu kullanırız, response.meta üzerinden screenshot'a ulaşılabilir ve f.write ile de ilgili screenshot png olarak proje içine kaydedilebilir.

Sonuçta bu spider çalıştırıldığında duckduckgo anasayfasının ss'i bir png file olarak elde edilmiş olacak.

## Spider'a giriş yaptık biraz ilerleyelim – duckduckgo :

```
import scrapy
from scrapy_selenium import SeleniumRequest
from selenium.webdriver.common.keys import Keys

class ExampleSpider(scrapy.Spider):
    name = 'example'

    def start_requests(self):
        yield SeleniumRequest(
            url='https://duckduckgo.com',
            wait_time=3,
            screenshot=True,
            callback=self.parse
        )

    def parse(self, response):
        # img = response.meta['screenshot']

        # with open('screenshot.png', 'wb') as f:
        #     f.write(img)

        driver = response.meta['driver']
        search_input = driver.find_element_by_xpath("//input[@id='search_form_input_homepage']")
        search_input.send_keys('Hello World')

        search_input.send_keys(Keys.ENTER)

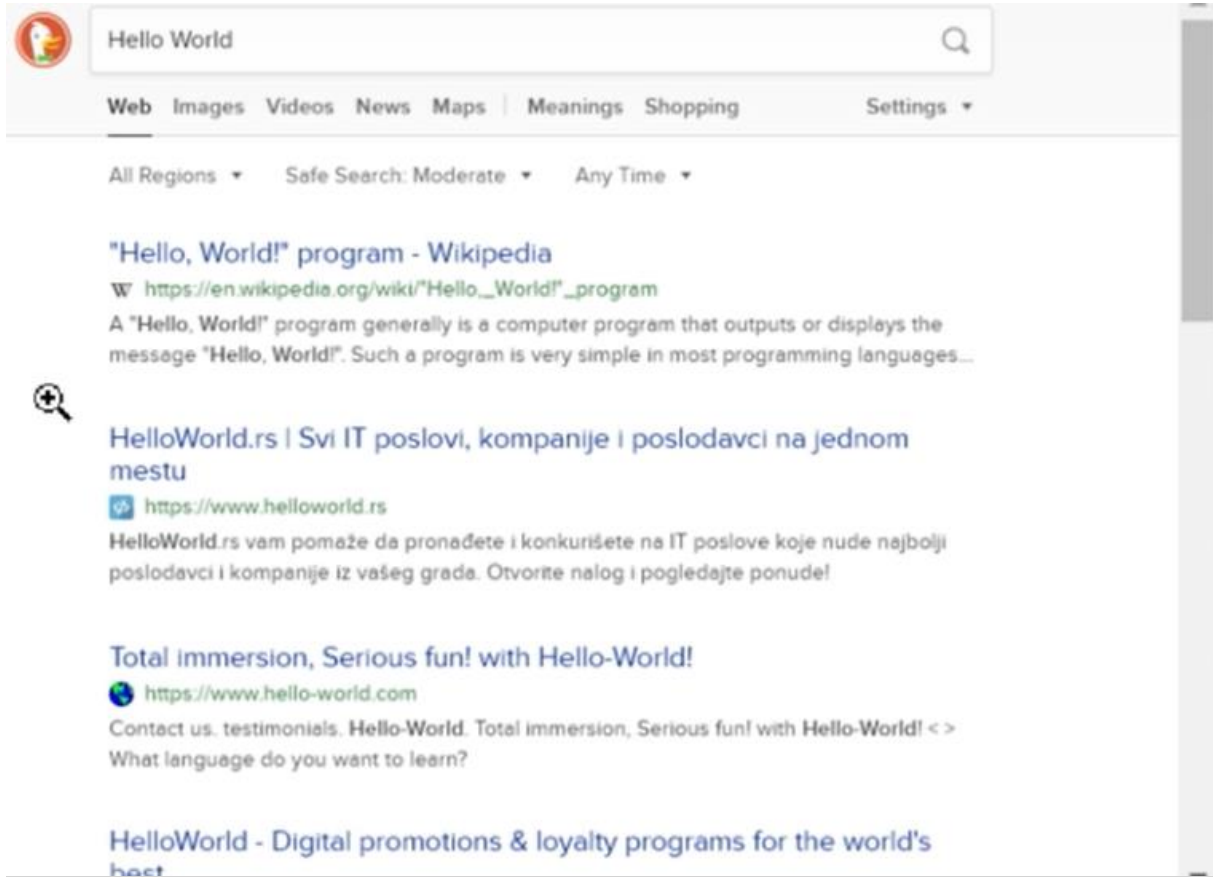
        driver.save_screenshot('enter.png')
```

Şimdi yapmak istediğimiz duckduckgo sayfasına gidip, input elemanı içerisine Hello World yazmak ve ENTER'a basarak karşımıza çıkan ekranın screenshotını almak:

- Bunu yapmak için initial response içerisinden driver'ı grab ediyoruz, artık bu driver'ı kullanarak current response üzerinde eleman seçme form doldurma vesaire işlerini daha önce yaptığımız gibi yapabiliriz.
- Burada da xpath ile input elemanı seçilmiş, içerisine Hello World yazılmış ve daha sonra ENTER'a basılmış.
- Son olarak screenshot almak için save\_screenshot() methodu kullanılmış.

Response.meta ile screenshot'a ulaşma olayı sadece initial request durumunda kullanılıyor, onun dışında save\_screenshot() kullanıyoruz.

Sonuçta aşağıdaki gibi bir screenshot dosyasını proje klasörüne kaydetmiş olacağız:



## Biraz daha ilerleyelim ve duckduckgo örneğini tamamlayalım:

Şimdi yapmak istediğim şey, Hello World yazdıktan sonra arama sonucunda karşımıza çıkan sonuçların her birinin url'ini elde etmek.

```
import scrapy
from scrapy.selector import Selector
from scrapy_selenium import SeleniumRequest
from selenium.webdriver.common.keys import Keys

class ExampleSpider(scrapy.Spider):
    name = 'example'

    def start_requests(self):
        yield SeleniumRequest(
            url='https://duckduckgo.com',
            wait_time=3,
            screenshot=True,
            callback=self.parse
        )

    def parse(self, response):
        # img = response.meta['screenshot']

        # with open('screenshot.png', 'wb') as f:
        #     f.write(img)

        driver = response.meta['driver']
        search_input = driver.find_element_by_xpath("//input[@id='search_form_input_homepage']")
        search_input.send_keys('Hello World')

        search_input.send_keys(Keys.ENTER)

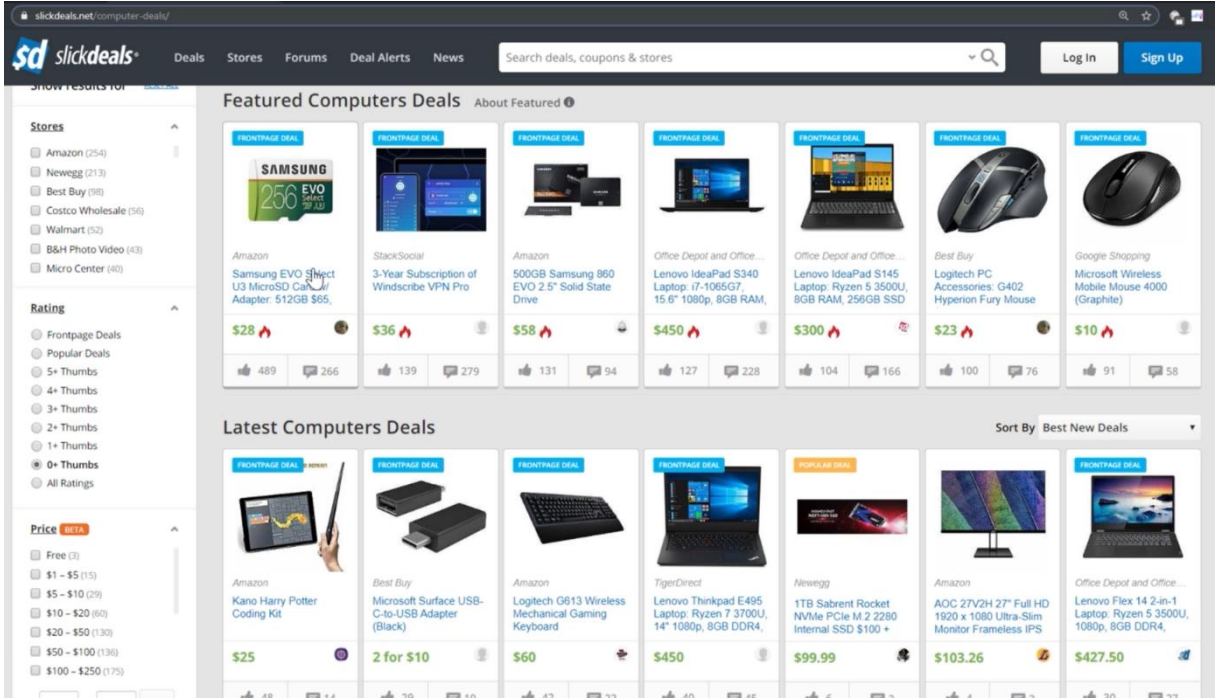
        html = driver.page_source
        response_obj = Selector(text=html)

        links = response_obj.xpath("//div[@class='result_extras_url']/a")
        for link in links:
            yield {
                'URL': link.xpath("//@href").get()
            }
```

- Parse içerisinde tanımlı response objesi, initial request'in response'u.
- Bizim linklerini scrape etmek istediğimiz sayfa ise Hello World arama sayfası, bu arama sayfasının ss'ini elde ettik ama response'u henüz elimizde yok bu sayfanın response'unu elde etmek için:
- Driver.page\_source methodunu kullanıyoruz ancak bu yeterli değil, bu response'u üzerinde xpath çalıştırılabilecek forma çevirmek için import edilen Selector modülünü kullanıyoruz.
- Son olarak da yukarıda görüldüğü gibi elde edilen response\_obj üzerinde xpath çalıştırarak her bir search elemanının linkini elde ediyorum.

# Selenium – Selenium Middleware – Project

Şimdi slickdeals.net sayfasını scrape edeceğiz. Bu sayfa aslında javascript ile çalışmıyor yani scrapy kullanmak yeterli olacak ancak pratik amaçlı sanki selenium'a ihtiyacımız varmış gibi düşünelim selenium ile scraping yapalım.



Amacımız her product için store image'ı, product name'i, store'u, url'i scrape edeceğiz.

Bu amaçla önceki başlıktaki proje içerisinde yeni bir spider yarattık, settings.py aynı kaldığı için configurations zaten yapıldı.



```

# -*- coding: utf-8 -*-
import scrapy
from scrapy_selenium import SeleniumRequest

class ComputerdealsSpider(scrapy.Spider):
    name = 'computerdeals'

    def remove_characters(self, value):
        return value.strip('\xa0')

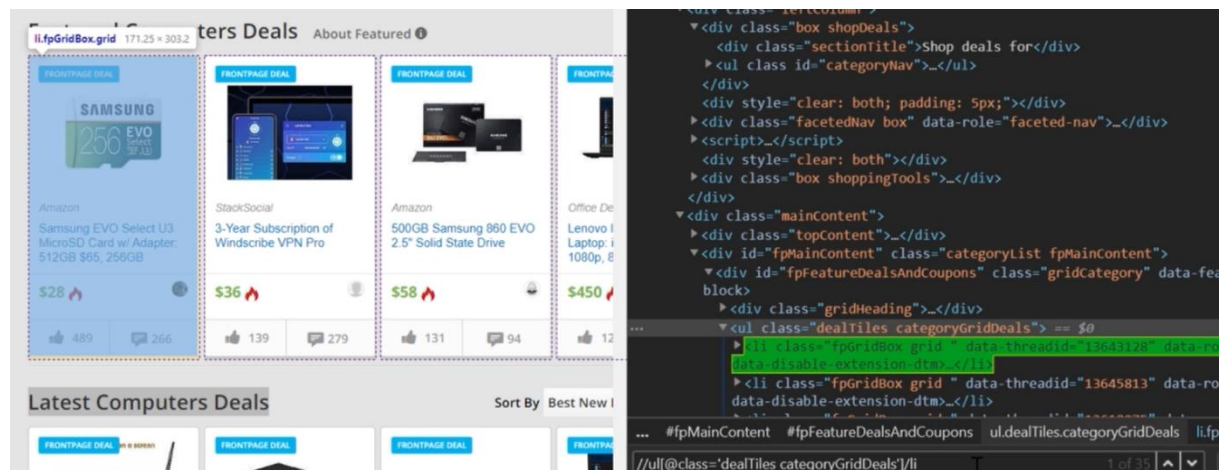
    def start_requests(self):
        yield SeleniumRequest(
            url='https://slickdeals.net/computer-deals/',
            wait_time=3,
            callback=self.parse
        )

    def parse(self, response):
        products = response.xpath("//ul[@class='dealFiles categoryGridDeals']/li")
        for product in products:
            yield {
                'name': product.xpath("//a[@class='itemTitle']/text()").get(),
                'link': product.xpath("//a[@class='itemTitle']/@href").get(),
                'store_name': self.remove_characters(product.xpath("normalize-space(//span[@class='itemStore']/text())").get()),
                'price': product.xpath("normalize-space(//div[@class='itemPrice wide']/text())").get()
            }

        next_page = response.xpath("//a[@data-role='next-page']/@href").get()
        if next_page:
            absolute_url = f"https://slickdeals.net{next_page}"
            yield SeleniumRequest(
                url=absolute_url,
                wait_time=3,
                callback=self.parse
            )

```

- Selenium request kullanılacağı için **start\_urls** ve **allowed\_domain** listeleri silindi.
- Start\_request methodu override edildi, sitenin computer-deals kısmına initial request gönderilsin dedik, response'u render etmeden önce 3 saniye bekle ve parse methodunu çağır dedik.
- Daha sonra parse içerisinde ilgili product bilgilerini içeren list elemanlarını seçip products içerisine kaydediyorum, products içerisinde 33 eleman yani 33 product elde ediliyor, daha sonra bu products içerisinde for loop ile dönüp her bir product için istediğim özellikleri yield ediyorum.



- **Store\_name** ve **price** içerisinde **normalize-space** kullandığımıza dikkat et, bunun sebebi bunu kullanmazsak output formatında gereksiz space'ler görüyor olmamız.
- Ayrıca aşağıda görüldüğü gibi store name içerisinde gereksiz bir **\xa0** yer alıyor. Bunu kaldırmak için de **remove\_characters** methodunu kullanıyoruz, bu methodu kendimiz tanımladık, ve store\_name'i yield etmeden önce çağırdık.

```
2019-12-09 14:45:14 [scrapy.core.scrapers] DEBUG: Scraped from <200 https://slickdeals.net/computer-deals/>
{'name': '32GB (2x 16) Geil-EVO POTENZA AMD DDR4-3000 Desktop RAM @Newegg $92.99', 'link': '/ff11702709-32gb-2x-16-geil-evo-potenza-amd-ddr4-3000-desktop-ram-newegg-92-99?src=atpagev2', 'store_name': 'Newegg\\xa0', 'price': '\n          $92.99\n'}
```

### - **Pagination'ı halletmek için ne yaptık?**

- İlgili sayfadaki 33 elemanı scrape ettik sırada, next page'e geçip o sayfayı da scrape etmek var.
- Öncelikle response.xpath kullanarak ilgili sayfadaki next\_page butonunun var olup olmadığını kontrol ediyoruz.
- Eğer böyle bir eleman varsa bu next\_page var demek o halde doğrudan next page url'ini oluşturuyoruz ve daha sonra SeleniumRequest ile yeni oluşturulan url'e request'imizi gönderip callback'i parse olarak atıyoruz.
- Sanıyorum scraping api kısmında bu mantığı kullanmıştık çünkü next butonuna basınca gidilecek sayfa bize yeni veriyi getirmiyordu onun yerine api request url'sini elde etmemiz gerekiyordu. Sonuçta next butonunun olmadığı veya scrapy ile next butonuna basılamadığı javascript tabanlı sayfalarda böyle bir yaklaşım kullanılabilir. Burada next butonu olduğu için ve selenium kullandığımız bence butona tıklamak da mantıklı.