

# Splash Crash Course

## Problem that splash solves

Javascript ile çalışan website içeriklerini scrapy ile çekemiyoruz, hatırlarsan bunu test etmek için önce browser'dan javascript'i disable ediyorduk ve websitesini öyle keşfediyorduk, planımızı öyle yapıyorduk.

Javascript client tarafında çalıştırılan kodlardır, yani javascript'i çalıştıran browser'ımızdır.

Şimdi problemi anlamak için aşağıdaki siteye bakabiliriz:

<https://www.livecoin.net/en>

Burada cryptocurrency anlık bilgileri yer alıyor, ancak ilk sayfada atıyorum 50 tane currency yer alıyor, ve pagination yerine show more butonu var bu show more butonu da javascript ile çalışıyor. Dolayısıyla ben bu sayfayı scrapy ile scrape etmeye çalıştığımda pagination yapamayacağım, sadece ilk 50 veriye ulaşabileceğim.

Bunun sebebi scrapy'nin javascript'i interpret edebilecek built-in-browser'a ya da engine'e sahip olmamasıdır.

Bu noktada bu problemi çözmek için ya **Splash** kullanabiliriz veya **Selenium** kullanabiliriz.

Splash is like a lightweight browser, bu browser'la interact etmek için chrome'daki gibi icon'lara tıklamayız bunun yerine coding ile bu browser ile interact ederiz. Ayrıca splash scrapy ile kullanılır.

Selenium ise test/automation tool web scrapping için develop edilmemiş, bu yüzden çok kullanmadığını söylüyor. Ancak beginner-friendly olduğu için açıklanacak sanıyorum.

Dediğimiz gibi javascript'in execute edilebilmesi için bir engine'e ihtiyacı var:

- Chrome uses V8 Engine kullanır.
- Firefox Spider Monkey kullanır.
- Safari Apple Webkit kullanır. Splash de bunu kullanır.
- Microsoft da Chakra kullanır.

## Setup Splash

Videoya bakarak kurulumu yapabilirsin, kurulumu yaptıktan sonra splash'ı çalıştırmak için cmd'ye:

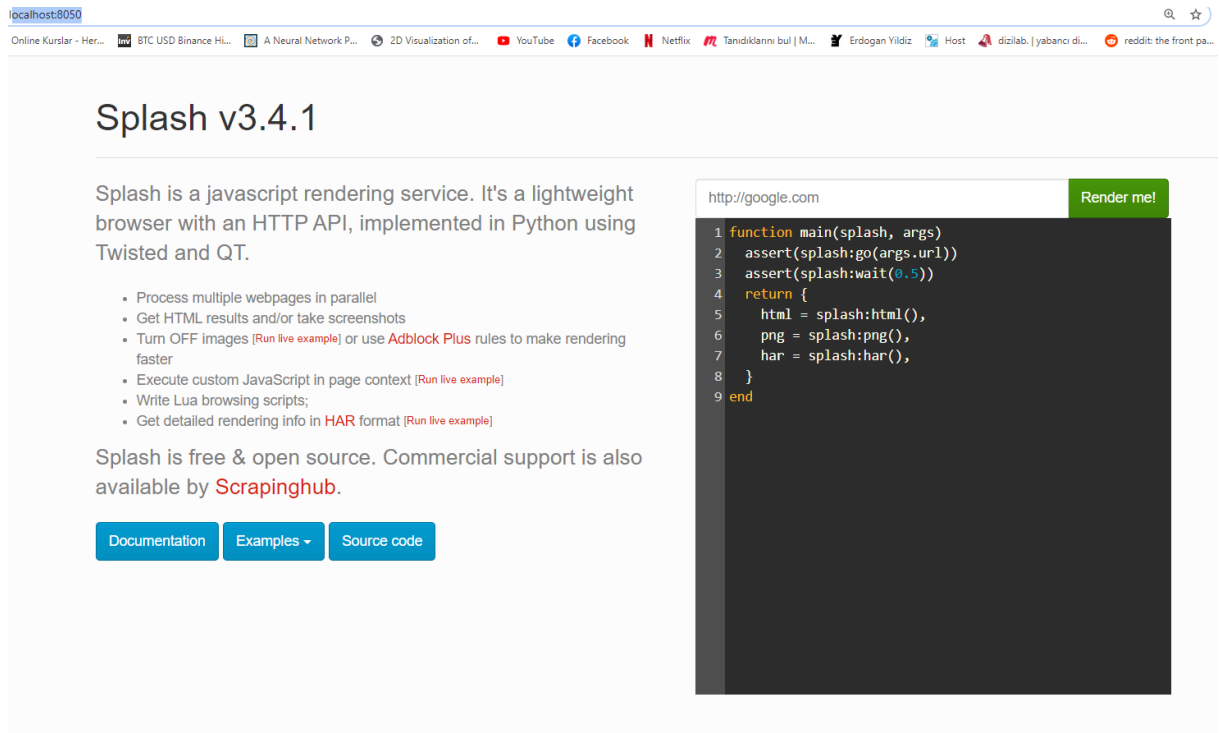
`docker run -it -p 8050:8050 scrapinghub/splash` dedik ve aşağıdaki gibi infolar aldık.

```
docker: /usr/bin/docker: Executing command: docker run -it -p 8050:8050 scrapinghub/splash
020-06-06 09:48:56+0000 [-] Log opened.
020-06-06 09:48:56.962902 [-] Xvfb is started: ['Xvfb', ':339873120', '-screen', '0', '1024x768x24', '-nolisten', 'tcp']
StandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-splash'
020-06-06 09:48:57.266402 [-] Splash version: 3.4.1
020-06-06 09:48:57.374715 [-] Qt 5.13.1, PyQt 5.13.1, WebKit 602.1, Chromium 70.0.3683.105, sip 4.19.19, Twisted 19.7.0, Lua 5.2
020-06-06 09:48:57.376889 [-] Python 3.6.9 (default, Nov 7 2019, 10:44:02) [GCC 8.3.0]
020-06-06 09:48:57.377643 [-] Open files limit: 1048576
020-06-06 09:48:57.378239 [-] Can't bump open files limit
020-06-06 09:48:57.399399 [-] proxy profiles support is enabled, proxy profile path: /etc/splash/proxy-profiles
020-06-06 09:48:57.399824 [-] memory cache: enabled, private mode: enabled, js cross-domain access: disabled
020-06-06 09:48:57.555863 [-] verbosity=1, slots=20, argument_cache_max_entries=500, max_timeout=90.0
020-06-06 09:48:57.556292 [-] Web UI: enabled, Lua: enabled (sandbox: enabled)
020-06-06 09:48:57.557110 [-] Webkit: enabled, Chromium: enabled
020-06-06 09:48:57.557312 [-] Site starting on 8050
020-06-06 09:48:57.557312 [-] Starting factory <twisted.web.server.Site object at 0x7fb1b6a76160>
020-06-06 09:48:57.557839 [-] Server listening on http://0.0.0.0:8050
```

İlgili adresin 8050 portunu server listening diyor.

Şimdi browser'ı açıyoruz ve

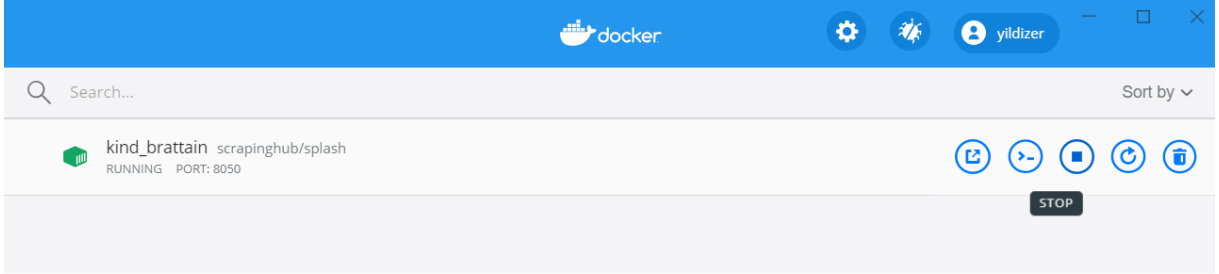
`localhost:8050` url'sine gidiyoruz, sonuç aşağıdaki gibi:



Böylelikle splash'ı run etmiş olduk. `Splash'ı durdurmak için de command window'da iki kez CTRL+C yaparım!`

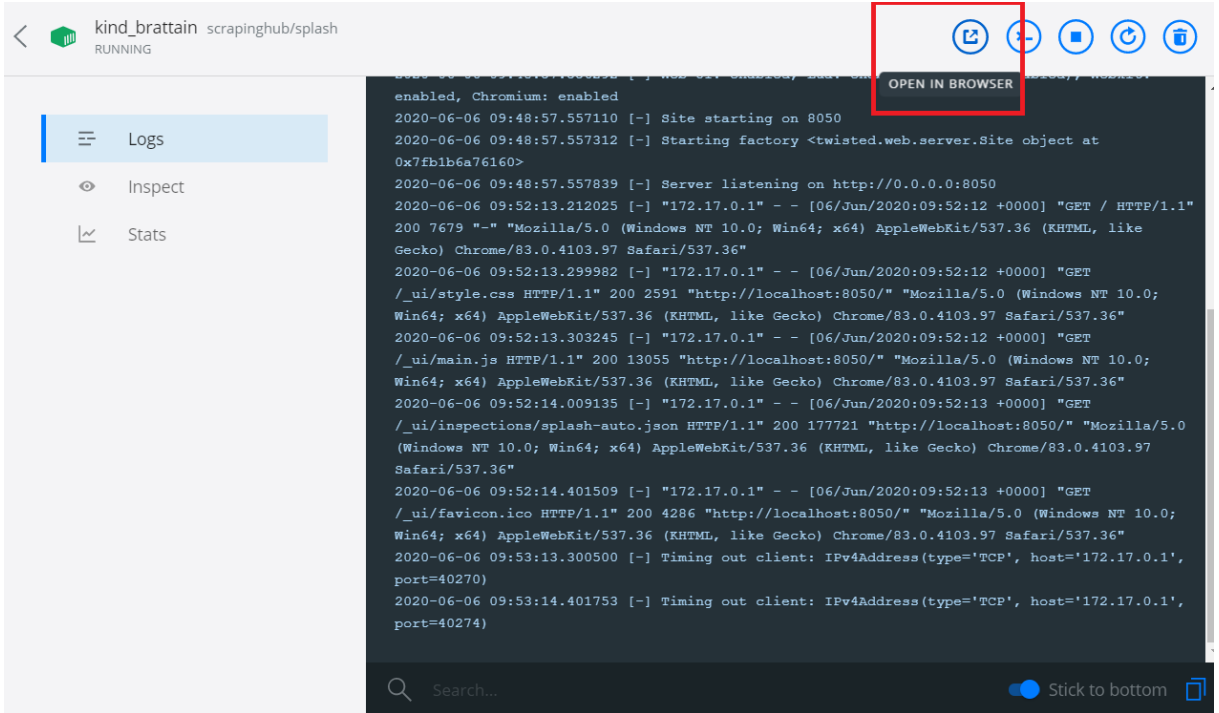
Ancak splash'i bir daha run etmek istediğimde her seferinde cmd'ye yukarıdaki komutları girmek zorunda değiliz.

Masaüstü Sağ alttan Docker Desktop'a right click → Dashboard deriz:



Yukarıdaki gibi bir ekran açılacak kind\_brattain isminde bir splash container görüyoruz, buradan ilgili splash container'ın çalışmasını durdurabiliriz, zaten daha önce durdurduysak da yeniden başlatabiliriz.

Logları görmek Start dedikten sonra, kind\_brattain kısmına tıklarız:



Buradan da splash'i browser'da açmak istersek yukarıda gösterilen ikona tıklayarak bunu yapabiliriz.

## Introduction to Splash

Localhost'da açılan splash user interface'i aşağıda görüldüğü gibi:

### Splash v3.4.1

Splash is a javascript rendering service. It's a lightweight browser with an HTTP API, implemented in Python using Twisted and QT.

- Process multiple webpages in parallel
- Get HTML results and/or take screenshots
- Turn OFF images [Run live example] or use **Adblock Plus** rules to make rendering faster
- Execute custom JavaScript in page context [Run live example]
- Write Lua browsing scripts;
- Get detailed rendering info in **HAR** format [Run live example]

Splash is free & open source. Commercial support is also available by [Scrapinghub](#).

[Documentation](#)[Examples ▾](#)[Source code](#)

http://google.com

Render me!

```
1 function main(splash, args)
2   assert(splash:go(args.url))
3   assert(splash:wait(0.5))
4   return {
5     html = splash:html(),
6     png = splash:png(),
7     har = splash:har(),
8   }
9 end
```

Sağ üstteki kısma target website url yazılır, daha sonra execute edilecek kod sağdaki kod bloğuna girilir ve render me! Butonu ile ilgili kodlar uygulanır.

Anladığım kadarıyla target website url'sini oraya yazmamız demek splash'ın render dediğimizde o siteye gideceği anlamına gelmiyor sanki bir değişken içine değer atamışız gibi düşün, yazılan adrese gitmesini yine kod içinde söylememiz gerek.

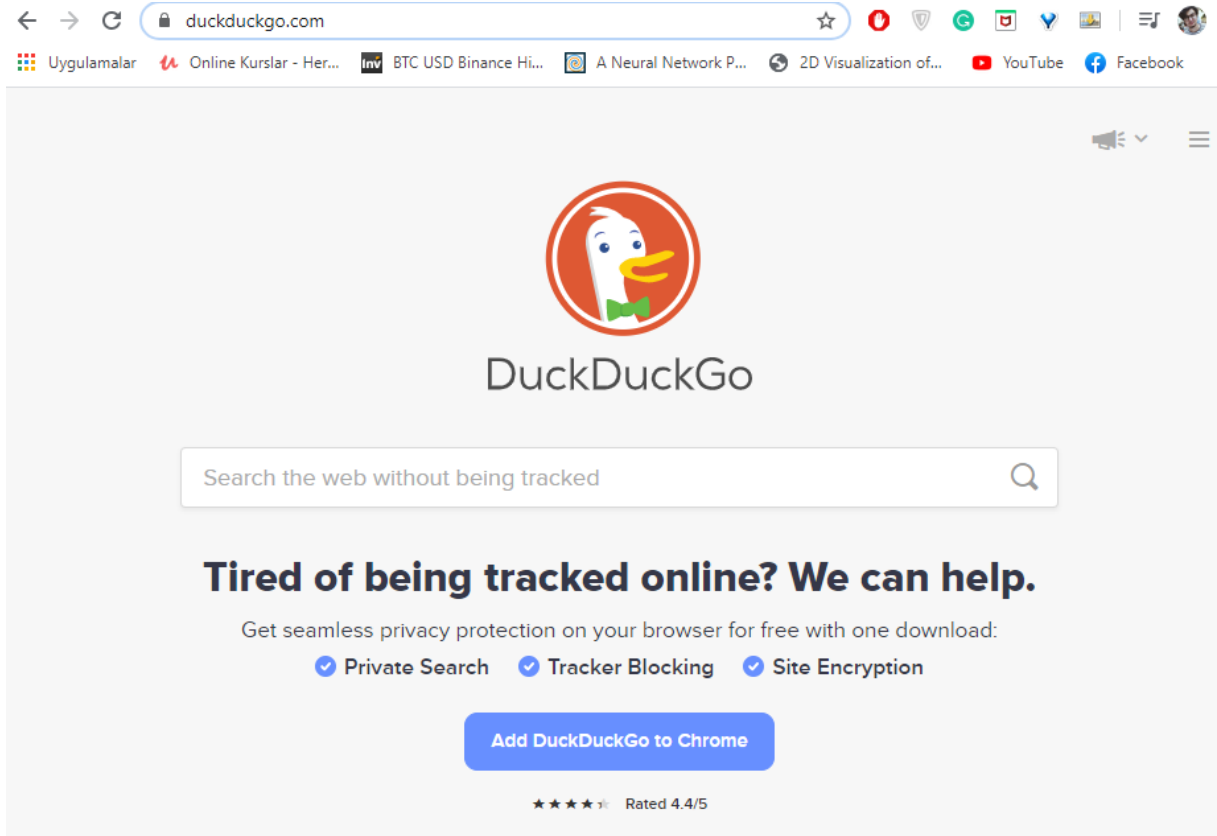
Daha önce de dediğimiz gibi splash bir browser gibidir, bu browser'la kod aracılığı ile interact ederiz, iconlarla değil.

Yukarıda yazılan örnek kod Lua dili ile yazılmış, ama bize zaten çok derinlemesine bir Lua bilgisi gerekmeyecek, yolda halledilir.

Ancak gerekli olursa şu linkten 15 dakikada Lua öğrenebilirsiniz:

<http://tylernelson.com/a/learn-lua/>

Şimdi biz splash'i anlamak için örneğimize geçelim, [duckduckgo.com](https://duckduckgo.com) sitesini örnek olarak kullanacağız:



Bu site'yi render etmek için splash kullanacağız ve bu süreçte şunları öğreneceğiz:

- **How to fulfill forms.**
- **How to perform mouse clicks.**
- **How to send keyboard keys.**
- **How to spoof request headers, custom ways of change request headers.**

## Intro - taking a screenshot of the webpage

Öncelikle Splash kullanarak ilgili websayfasının screenshot'ını alacağız.

Öncelikle url kısmına ilgili sitenin url'sini giriyoruz daha sonra kodu yazacağız, ancak direkt yazılan url'den response alınıyor ve kod onun üzerinde çalışıyor gibi düşünme, render me diyince sadece kod çalışıyor o kadar, hangi url'ye gideceğini de biz söyleyeceğiz.

Elbette kutucuğa yazılan url'yi direct kod içinde çekebiliyoruz.

Önce bir main function oluşturuyoruz, bu main function iki argüman alacak splash ve args.

- Fonksiyon içerisinde öncelikle yukarı yazılan target website url'sini elde ediyoruz.
- Bu amaçla `url = args.url` diyerek girilen url'i string olarak url değişkeni içerisine kaydettik.
- Daha sonra bu url'ye gitmek için `splash:go(url)` fonksiyonunu kullanıyoruz.
- Son olarak ziyaret edilen website'ın screenshot'ını almak için de `return splash:png()`

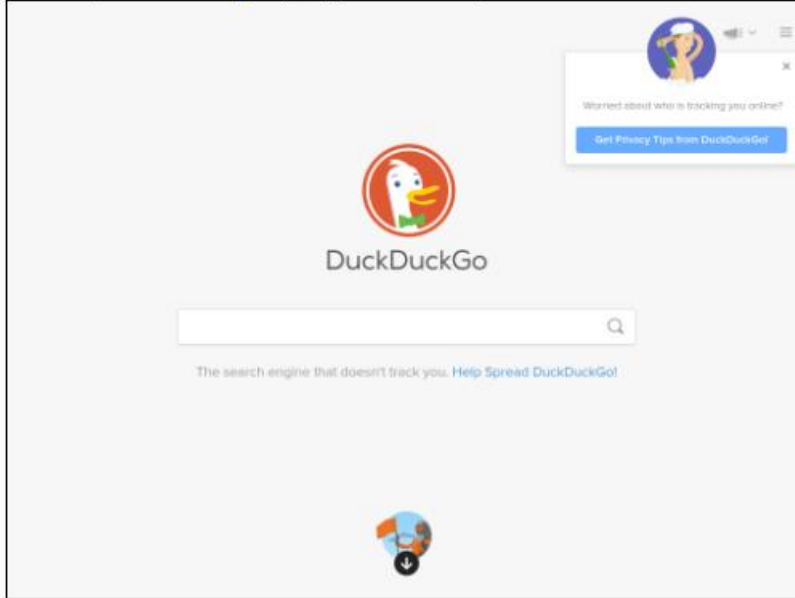
`https://duckduckgo.com/`

Render me!

```
1 function main(splash,args)
2   url=args.url
3   splash:go(url)
4   return splash:png()
5 end
```

Yukarıda yazılan kodu render edersek, splash'ın response'unu görebiliriz:

Splash Response: Image (png, 1024x768) [download](#)



### Taking a HTML markup of the webpage

Yukarıdaki kodda çok basit bir değişiklik yaparak screenshot yerine sitenin html markup'ını elde edebilirim:

`splash:png()` fonksiyonu yerine `splash:html()` fonksiyonunu kullanmam yeterli olacaktır. Sitenin html markup'ı string olarak döndürülecektir.

Success

<https://duckduckgo.com/>

Script ▾

```
1 function main(splash,args)
2   url=args.url
3   splash:go(url)
4   return splash:html()
5 end
```

## Hem html markup'ı hem de screenshot'u return ettirmek:

Bunun için iki elemanı bir object içinde wrap edip return ettiririz.

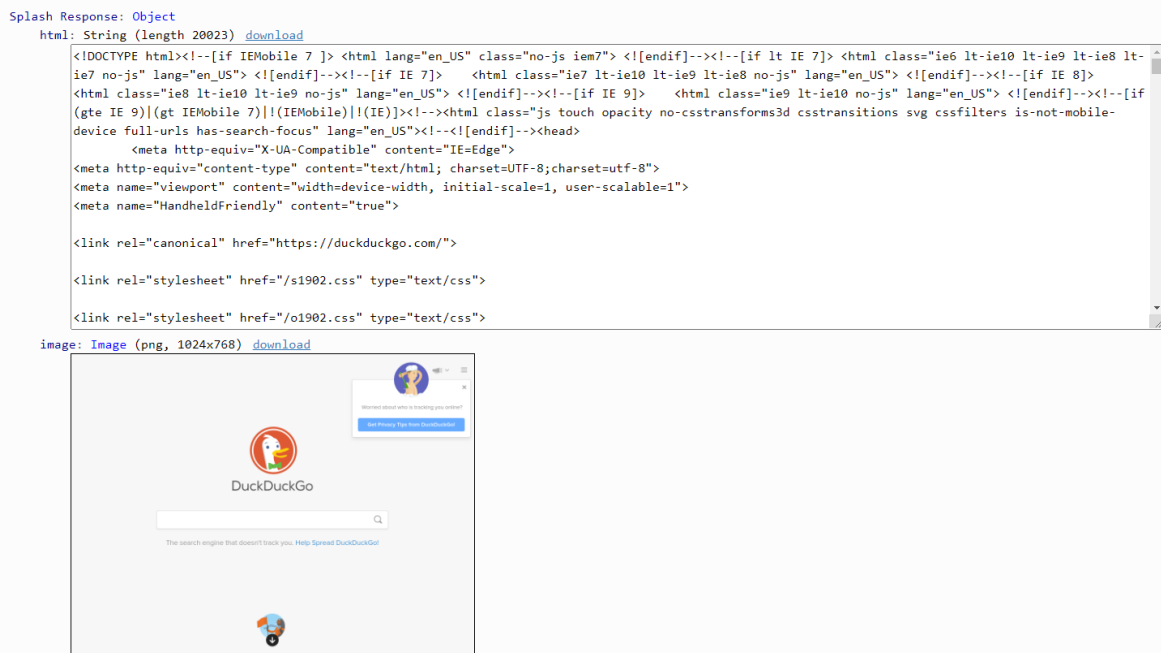
Lua'da obje oluşturmak aşağıdaki gibi oluyor, dictionary gibi:

```
{ image=splash.png(),  
  html=splash.html() }
```

Sonuçta bu objeyi return ettireceğim, yani kod aşağıdaki gibi olacak:

```
1 function main(splash,args)  
2   url=args.url  
3   splash:go(url)  
4   return {  
5     image = splash.png(),  
6     html = splash.html()  
7   }  
8 end
```

Bu objeyi return ettirdiğim zaman aşağıdaki gibi iki sonucu da object içerisinde görebilirim:

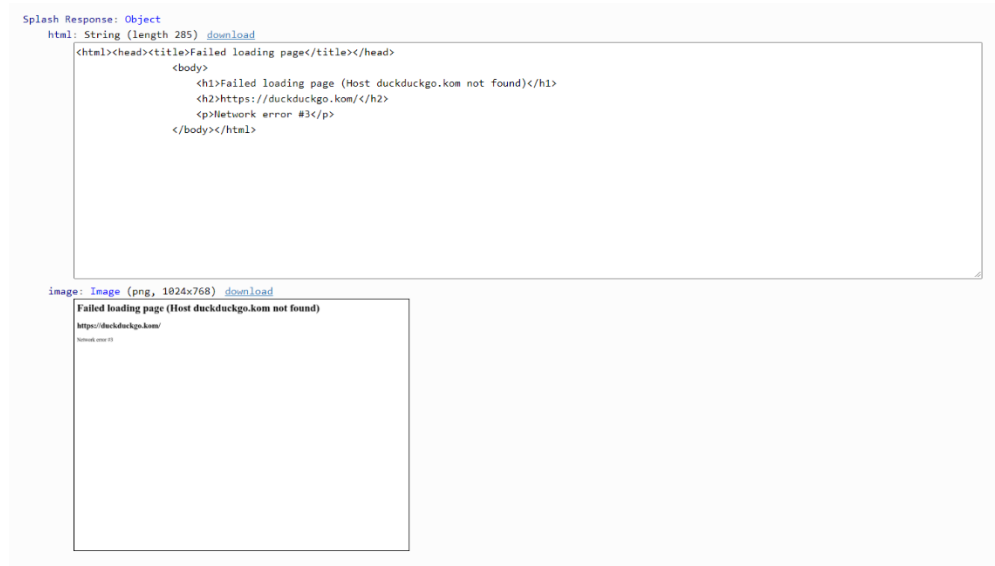




## Peki ya Invalid bir URL ismi girilmişse:

Mesela `duckduckgo.com` adresine ulaşmaya çalışıyoruz diyelim invalid url.

Bir önceki section'daki kodu çalıştırırsak, göreceğiz ki splash bu sayfaya ulaşmaya çalışıyor, ve bu sayfaya ulaşamasa da yönlendirildiği hata sayfası üzerinde söylenen işlemleri yapıyor, yani markup'ı screenshot'ı alıyor.



Yani aslında `splash:go(url)` komutu url'i açamadı, failed ama yine de kalan return komutu çalıştı.

```
1 function main(splash,args)  
2   url=args.url  
3   splash:go(url)  
4   return {  
5     image = splash:png(),  
6     html = splash:html()  
7   }  
8 end
```

Bu iyi bir practice değil, eğer açmaya çalıştığımız url açılmıyorsa, biz bunu bir hata olarak almak isteriz ve kodun da durmasını isteriz.

Bunu başarmak için go satırına assert ekliyoruz, yani eğer splash:go(url) true ise yani işlem başarılı ise hiçbir şey olmasın, yok false ise yani adrese ulaşamamışsa assertion error döndürülsün.

```
1 function main(splash,args)
2   url=args.url
3   assert(splash:go(url))
4   return {
5     image = splash:png(),
6     html = splash:html()
7   }
8 end
```

Sonuçta aşağıdaki gibi bir Bad Request hatası alıyoruz ve hatanın hangi satırda olduğu vesaire gibi bilgileri görüyoruz.

HTTP Error 400 (Bad Request)

Type: ScriptError -> LUA\_INIT\_ERROR

Error happened while executing Lua script

[string "function main(splash,args) ..."]:2: ')' expected near '='

```
{
  "error": 400,
  "type": "ScriptError",
  "description": "Error happened while executing Lua script",
  "info": {
    "source": "[string \"function main(splash,args)\\r...\\\"]",
    "line_number": 2,
    "error": "')' expected near '='",
    "type": "LUA_INIT_ERROR",
    "message": "[string \"function main(splash,args)\\r...\\\"]:2: ')' expected near '='"
  }
}
```

### Wait after visiting a URL:

Bir url ziyaret edildikten sonra, 0.5 veya 1 saniye gibi bir süre beklemek good practice'dir. Böylece splash'e resource'u yüklemesi için yeterli süreyi veriyoruz.

Bu örnekte bu beklemeye gerek duymadık çünkü duckduckgo'nun yüklenmesi yeterince hızlıydı, ancak genelde go işleminden sonra biraz beklemek önerilir:

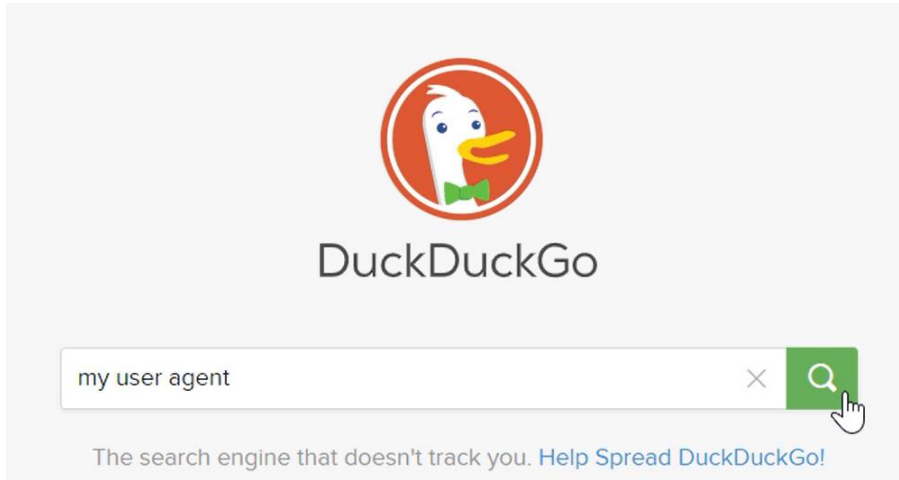
```
1 function main(splash,args)
2   url=args.url
3   assert(splash:go(url))
4   assert(splash:wait(1))
5   return {
6     image = splash:png(),
7     html = splash:html()
8   }
9 end
```

Böylece return statement'ı execute edilmeden önce 1 saniye beklenecek.

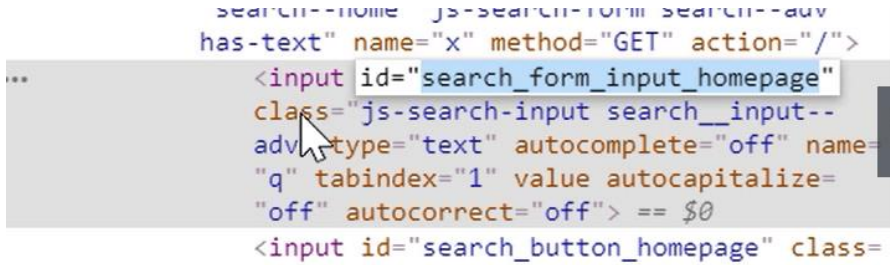
## Filling in Forms and Clicking in Splash

### Filling an input element

İlk yapmak istediğimiz şey, duckduckgo anasayfasındaki forma bir girdi("my user agent") girmek ve daha sonrasında da search butonuna tıklamak veya enter'a basmak.



Inspector ile form elemanına baktığımızda bunun search\_form\_... şeklinde bir id'si olan bir input elemanı olduğunu anlıyoruz:



Şimdi splash koduna dönüyorum:

splash:go(url) ile url'yi visit ettikten ve 1 saniye bekledikten sonra

```
input_box = assert(splash:select("#search_form_input_homepage"))
```

satırını ekledik, burada yapılan şey ilgili ID'ye sahip elemanı sayfanın html markup'ından seçmek, yani input elemanı seçilecek. Ancak splash'ın select fonksiyonu sadece CSS Selector alıyor, xpath ile çalışmaz, zaten css selector öğrenmemizin tek sebebi buydu.

**select()** sadece tek bir eleman seçeceksek kullanılır, ancak eğer birden fazla eleman seçilecekse **select\_all()** kullanılır.

Seçilen elemanın içine text vereceğiz ancak bunu yapmak için önce ilgili elemanı focus state'e getirmemiz gerek bu yüzden:

`input_box:focus()` ile `input_box` elemanını focus state'e aldık.

Son olarak `input_box:send_text("my user agent")` komutu ile ilgili `input_box` elemanının içine "my user agent" text'ini yolluyoruz.

Fill işleminin yapılması için biraz zaman tanıyalım yani bir wait komutu koyalım ve sonra, önceki gibi ss'i ve markup'ı return edebiliriz.

`assert(splash:wait(0.5))`

Sonuçta kod aşağıdaki gibi oldu:

```
1 function main(splash, args)
2   url = args.url
3   assert(splash:go(url))
4   assert(splash:wait(1))
5
6   input_box = assert(splash:select("#search_form_input_h
7   input_box:focus()
8   input_box:send_text("my user agent")
9   assert(splash:wait(0.5))
10  return {
11    image = splash:png(),
12    html = splash:html()
13  }
14 end
```

Duckduckgo url'si açıldı, 1 saniye beklendi, input elemanı id ile seçildi, elemana focuslandı, text girildi, 0.5 saniye beklendi, ve ss ile html markup return edildi.

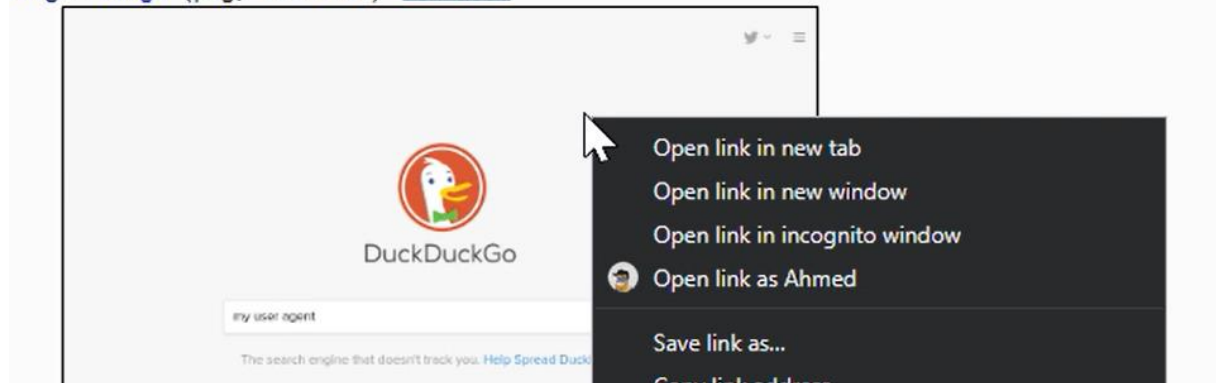
Sonuçta bakılırsa, elde edilen screenshot'da input elemanına istediğimiz text'in girdi olarak verildiğini görebiliyoruz:

```
lt-ie7 no-js" lang="en_US"> <![endif]--><!--[if IE 7]> <html class="ie7 lt-ie10 lt
<html class="ie8 lt-ie10 lt-ie9 no-js" lang="en_US"> <![endif]--><!--[if IE 9]> <h
[if (gte IE 9)|(gt IEMobile 7)|!(IEMobile)|!(IE)]><!--><html class="js touch opacity
mobile-device full-urls has-search-focus" lang="en_US"><!--<![endif]--><head><script
callback=autocompleteCallback&amp;q=my+user+agent&amp;kl=wt-wt&amp;_1566903249012"><
<meta http-equiv="X-UA-Compatible" content="IE=Edge">
<meta http-equiv="content-type" content="text/html; charset=UTF-8; charset=utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1, user-scalable=1">
<meta name="HandheldFriendly" content="true">

<link rel="canonical" href="https://duckduckgo.com/">

<link rel="stylesheet" href="/s1816.css" type="text/css">
```

image: Image (png, 1024x768) [download](#)



## Mouse Click to an Element

Search input'a değer girdik, şimdi yapacağımız şey search butonunu seçip tıklamak.

Input elemanını seçip, değer girmeye çok benzer olacak bu, öncelikle search butonunu inspect edeceğiz ve id'sini bulacağız.

Daha sonra:

```
btn = assert(splash:select("#search_button_homepage"))
```

```
btn:mouse_click()
```

```
assert(splash:wait(3))
```

 wait time'ı duruma göre artırıp azaltabiliriz.

Dedik yani buton elemanını seçtik ve mouse click operation'ını gerçekledikten sonra 3 saniye yeni sayfanın yüklenmesini bekledik.

Daha sonra yine html markup ile ss'i return ettireceğiz yani total kod şu hale geldi:

```
1 function main(splash, args)
2   url = args.url
3   assert(splash:go(url))
4   assert(splash:wait(1))
5
6   input_box = assert(splash:select("#search_form_input_h
7   input_box:focus()
8   input_box:send_text("my user agent")
9   assert(splash:wait(0.5))
10
11  btn = assert(splash:select("#search_button_homepage"))
12  btn:mouse_click()
13  assert(splash:wait(1))
14  return {
15    image = splash:png(),
16    html = splash:html()
17  }
18 end
```

Yukarıdaki kodu çalıştırınca sonuç aşağıdaki gibi oluyor:



Duckduckgo'da my user agent girdisi ile arama yaptık ve açılan sayfanın ss'ini aldık. Bu ss'in tüm sayfayı göstermediğine dikkat et, çünkü default olarak splash 1024x766 resolution kullanır, tüm sayfanın ss'ini elde etmek için splash:png()'den önce bir yere,

**splash:set\_viewport\_full()**

komutunu ekleriz.



## Press Enter Key

Şimdi aramayı search butonuna tıklayarak değil de inputu girdikten sonra, search input elementi seçiliyken Enter tuşuna basarak yapmak istiyoruz diyelim.

Bunun için buton tıklama kodlarını comment'e alıyorum. `input_box` elemanı eskisi gibi seçiliyor, focus ediliyor, içine "my user agent" text'i giriliyor, daha sonra

```
input_box:send_keys("<Enter>")
assert(splash:wait(5))
```

Yani sonuçta kod şu hale gelmiş oldu:

```
function main(splash, args)
  url = args.url
  assert(splash:go(url))
  assert(splash:wait(1))

  input_box = assert(splash:select("#search_form_input_
  input_box:focus()
  input_box:send_text("my user agent")
  assert(splash:wait(0.5))
  --[[
  btn = assert(splash:select("#search_button_homepage")
  btn:mouse_click()
  --]]

  input_box:send_keys("<Enter>")
  assert(splash:wait(5))
  return {
    image = splash:png(),
    html = splash:html()
  }
end
```



## Spoofing Request Headers

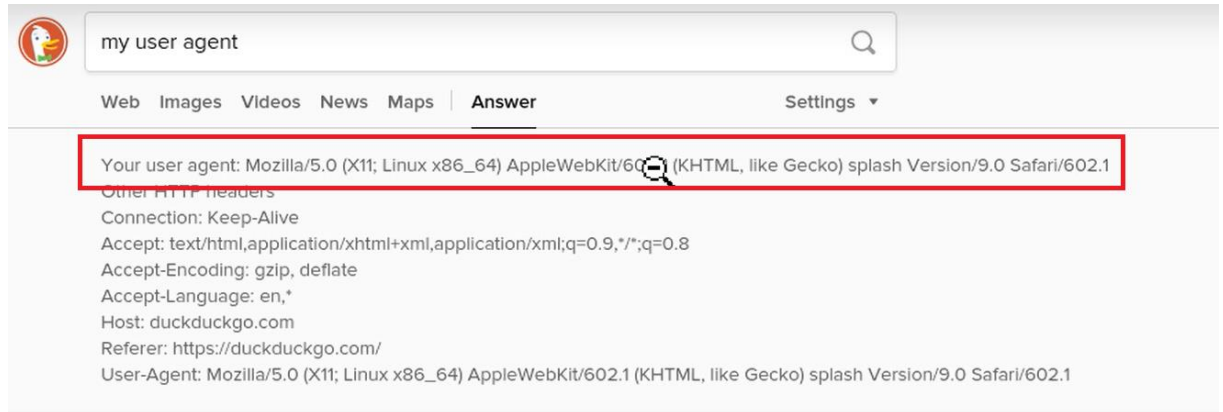
Daha önce siteden block yememek için, scrapy içerisinde birkaç farklı yolla user-agent request header'ı değiştirmeyi öğrenmiştik.

Bu başlık altında aynı işlemi Splash için nasıl yapabileceğimizi öğreneceğiz.

Şimdi splash'ın aynen chrome gibi bir browser olduğunu unutma daha doğrusu bir engine, chrome gibi tuşlarla kontrol edilemiyor onun yerine hangi sayfaya gidecek hangi sayfayı açacak onları bizim yazdığımız kodlara göre yapıyor.

Ancak bunun yaptığı işleri aynı bir browser yapıyormuş gibi düşün, aynen chrome'da sayfayı açıyoruz, tıklıyoruz, bekliyoruz vesaire.

Biz chrome'da google'a my user agent yazınca, chrome'un request gönderirken hangi user agent header'ını kullandığını elde etmiştik, benzer şekilde, splash üzerinden duckduckgo'ya my user agent yazdık ve aldığımız ss'den splash'ın user agent request header'ının değerini görebilirim:



The screenshot shows a DuckDuckGo search interface. The search bar contains the text "my user agent". Below the search bar, there are tabs for "Web", "Images", "Videos", "News", "Maps", and "Answer". The "Answer" tab is selected. The search results show a box with the text: "Your user agent: Mozilla/5.0 (X11; Linux x86\_64) AppleWebKit/602.1 (KHTML, like Gecko) splash Version/9.0 Safari/602.1". Below this box, there are several lines of text: "Other HTTP headers", "Connection: Keep-Alive", "Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8", "Accept-Encoding: gzip, deflate", "Accept-Language: en,\*", "Host: duckduckgo.com", "Referer: https://duckduckgo.com/", and "User-Agent: Mozilla/5.0 (X11; Linux x86\_64) AppleWebKit/602.1 (KHTML, like Gecko) splash Version/9.0 Safari/602.1".

All Regions ▼ Safe Search: Moderate ▼ Any Time ▼

### What is my User Agent? - WhatIsMyBrowser.com

<https://www.whatismybrowser.com/detect/what-is-my-user-agent>

Check out our **user agent** analyser page, which gives you a neat breakdown of all the things we can tell you about your browser and computer based on your **user agent**. We have a **User Agent** API if you need to use the detection in your own system too. Can I change my user agent? It is possible to change or "fake" what your web browser sends as its ...

### What's my user agent?

<https://www.whatsmyua.info>

Yukarıda görülen bu user-agent değeri, splash'ın default değeri, bizim burada yapacağımız şey ise bu değeri değiştirmek.

### First way

Main function içine en başa şunu yazıyoruz:

```
splash:set_user_agent("istediğimiz değeri buraya kopyalıyoruz, mesela kendi chrome broser'ımızın user agent değerini")
```

```
1 function main(splash, args)
2   splash:set_user_agent("Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.149 Safari/537.36")
3   url = args.url
4   assert(splash:go(url))
```

Bu yöntemle user-agent request header'ı override edebiliriz, peki ya diğer request headers'ı da override etmek istersek?

### Second Way

Bunu yapmak için ise bir object yaratabiliriz, ve bunun içinde istenilen tüm header'ları override edebiliriz aşağıda sadece user-agent override edilecek.

```
headers = {
```

```
  ['User-Agent'] = "istenilen value"
```

```
}
```

```
splash:set_custom_headers(headers)
```

Yukarıdaki kod yine main'in hemen içine yazılıyor:

```
1 function main(splash, args)
2   --splash:set_user_agent("Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.149 Safari/537.36")
3   headers = {
4     ['User-Agent'] = "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.149 Safari/537.36"
5   }
6   splash:set_custom_headers(headers)
7   url = args.url
8   assert(splash:go(url))
```

### Third Way

Yukarıdaki yöntemlere alternatif olarak, her request için çağırılacak bir callback function tanımlayabiliriz.

```
1 function main(splash, args)
2   --splash:set_user_agent("Mozilla/5.0 (Windows NT 10.
3   --[[
4   headers = {
5     ['User-Agent'] = "Mozilla/5.0 (Windows NT 10.0; Wi
6   }
7   splash:set_custom_headers(headers)
8   --]]
9   splash:on_request(function(request)
10     request:set_header('User-Agent', '')
11   end)
12   url = args.url
13   assert(splash:go(url))
14   assert(splash:wait(1))
```

Burada her request için request header'larından 'User-Agent' yanına girilecek value parametresi ile değiştirilecek.

Yukarıda value parametresi boş, buraya önceden kullanılan chrome user-agent değeri konulacak.

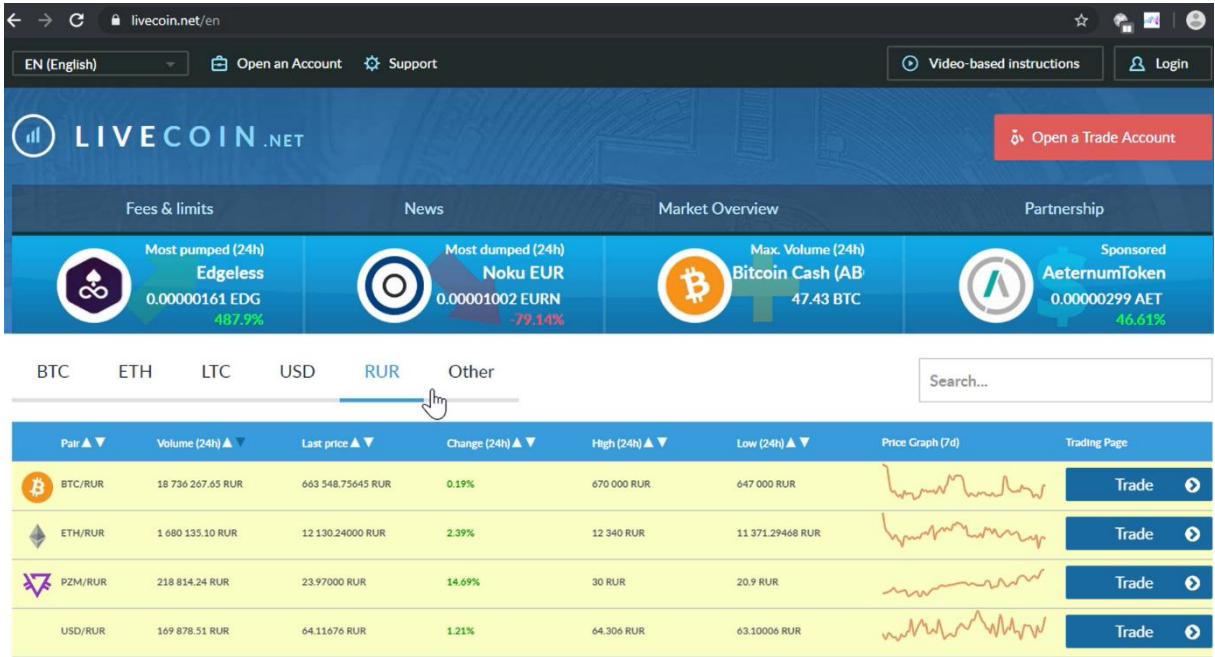
# Splash Project – Scraping Javascript Websites

## Intro

Şimdi örnek proje olarak [www.livecoin.net/en](http://www.livecoin.net/en) sitesini scrape edeceğiz, scrapy ile bu siteyi scrape edemiyoruz çünkü javascript interpretation gerekiyor ve scrapy kendi başına bunu yapamıyordu.

Sitede farklı currency'leri seçtiğimizde veya show more dediğimizde hiçbir şekilde url'in değişmediğini görüyoruz, yani butona basınca buton bizi başka bir url'ye götürmek yerine javascript ile işleri yürütüyor işte tam da bu yüzden splash'e ihtiyaç duyuyoruz.

İlgili sayfa aşağıda görüldüğü gibi bir şey biz burada RUR sekmesinden bilgileri çekeceğiz.



Ancak şuanki sayfada RUR kaldırılmış o zaman USD cinsinden verileri çekelim.

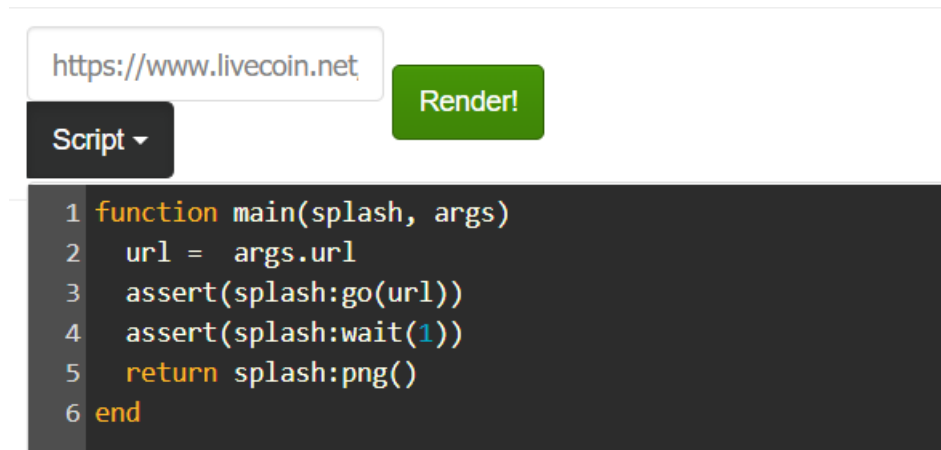
## Splash Incognito Mode

- Splash default olarak private/incognito mode'da bu durum scraping sürecinde bazı problemlere yol açabiliyor mesela, USD butonuna tıklasak da javascriptte bir değişim olmuyor vesaire.
- İşte bu yüzden default olarak enabled olan private mode'u disable etmek.
- Bu işlem için main function'ın hemen içine:  
`splash.private_mode_enabled = false` derim.
- Böylelikle garip bug'lardan kaçınırım.

## Splash Üzerinde Siteyi Scrape Etmek

Öncelikle siteyi sadece splash kullanarak scrape edelim, bunun için:

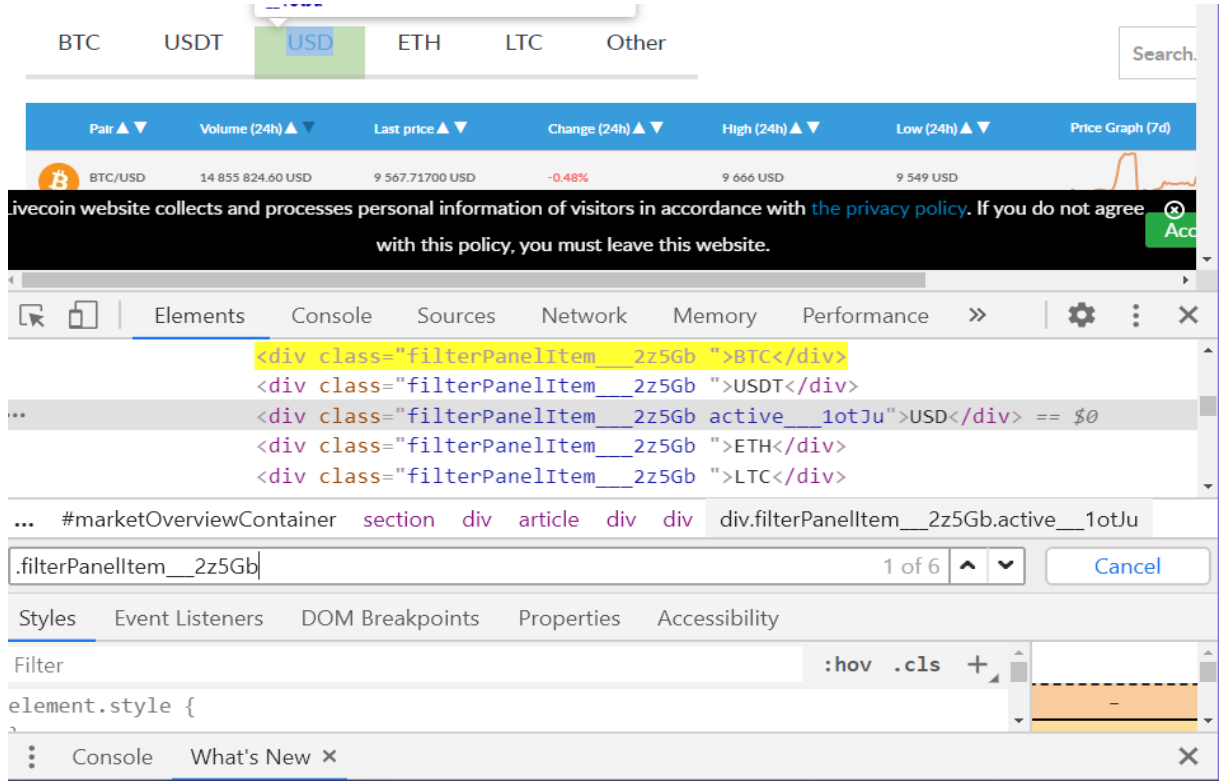
- URL'yi kopyala ve splash'ın input box'ına yapıştır.
- URL'yi al, visit et, 1 saniye bekle, screenshot'ı return et.



Burada `url=args.url`'nin hemen üstünde `splash.private_mode_enabled = false` satırının olması gerektiğini unutma. Yoksa incognito mode'dan dolayı problem yaşayabilirsin.

Bunu hallettik, sıradaki adımda USD elemanına tıklamak var, böylece BTC/USD, ETH/USD, LTC/USD oranlarını gösteren sayfa javascript ile elde edilecek.

- Öncelikle inspector ile USD elemanının css selector'ını tespit ediyoruz.



- Gördüğümüz gibi iki class'lı bir div elemanı, filterPanelItem class'ından 6 elemanın 3.sü olarak düşünülebilir ve biz de öyle seçeceğiz.
- Burada ilgili class'a sahip elemanları seçerken, splash içerisinde select\_all kullanacağız çünkü birden fazla eleman seçilecek. Daha sonra seçilen elemanların içinden kendi istediğimizi seçeceğiz.

İlgili class'a ait tüm elemanlar seçilsin.

```
usd_tab = assert(splash:select_all(".filterPanelItem__2z5Gb"))
```

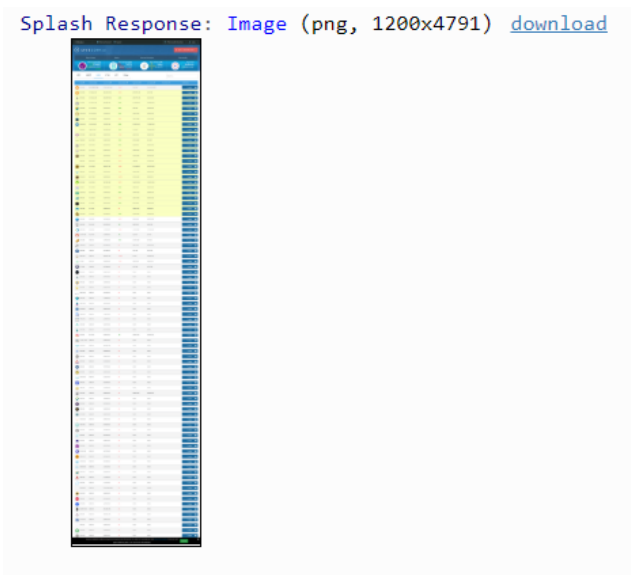
Seçilen elemanların 3.süne mouse click perform edilsin:

```
usd_tab[3]:mouse_click()
```

Sonuçta USD sekmesine tıklayıp full sayfanın ss'ini alan final kodumuz aşağıdaki gibi:

```
Success https://www.livecoin.net Script ▾  
1 function main(splash, args)  
2   splash.private_mode_enabled = false  
3   --url alınsın:  
4   url = args.url  
5   --ilgili url visit edilsin  
6   assert(splash:go(url))  
7   --1 saniye beklensin  
8   assert(splash:wait(1))  
9  
10  --usd elemanı seçilsin ve mouse ile tıklansın  
11  usd_tab = assert(splash:select_all(".filterPanelItem_  
12  usd_tab[3]:mouse_click()  
13  
14  --1 saniye beklensin  
15  assert(splash:wait(1))  
16  
17  --screenshot tüm ekranın fotoğrafını versin  
18  splash:set_viewport_full()  
19  
20  --screenshot return edilsin  
21  return splash:png()  
22 end
```

Çıktı da aşağıdaki gibi olacak:



Bu kısımda son olarak `splash:png()`'yi return etmek yerine `splash:html()`'i return ettik çünkü scrapy için html markup'a ihtiyacımız olacak.



## Using Splash with Scrapy

Öncelikle bir scrapy projesi ve spider'ı oluşturalım daha sonra splash'i scrapy ile nasıl kullanacağımızı anlayalım:

Projects klasörünün içinde livecoin isminde yeni bir proje yaratalım ve proje içersinde de coin isminde bir basic spider yaratalım:

```
(virtual_workspace) C:\Users\Ahmed\projects>scrapy startproject livecoin
New Scrapy project 'livecoin', using template directory 'C:\Users\Ahmed\Anaconda3\envs\virtual_w
ackages\scrapy\templates\project', created in:
  C:\Users\Ahmed\projects\livecoin

You can start your first spider with:
  cd livecoin
  scrapy genspider example example.com

(virtual_workspace) C:\Users\Ahmed\projects>cd livecoin

(virtual_workspace) C:\Users\Ahmed\projects\livecoin>scrapy genspider coin www.livecoin.net/en
Created spider 'coin' using template 'basic' in module:
  livecoin.spiders.coin

(virtual_workspace) C:\Users\Ahmed\projects\livecoin>
```

## Setting Up the Scrapy-Splash

Şimdi, scrapy projemizde splash'i kullanabilmek için SCRAPY-SPLASH isimdeki package'ı yüklememiz gerekecek.

Aşağıdaki github repository' de paketi nasıl yükleyeceğimiz scrapy ile nasıl configure edeceğimiz detaylıca anlatılmış. Ayrıca kullanım örnekleri de verilmiş, buradan detaylı bilgi edinebiliriz.

[github.com/scrapy-plugins/scrapy-splash](https://github.com/scrapy-plugins/scrapy-splash)

Şimdi öncelikle biz de bu package'ın kurulumunu yapacağız.

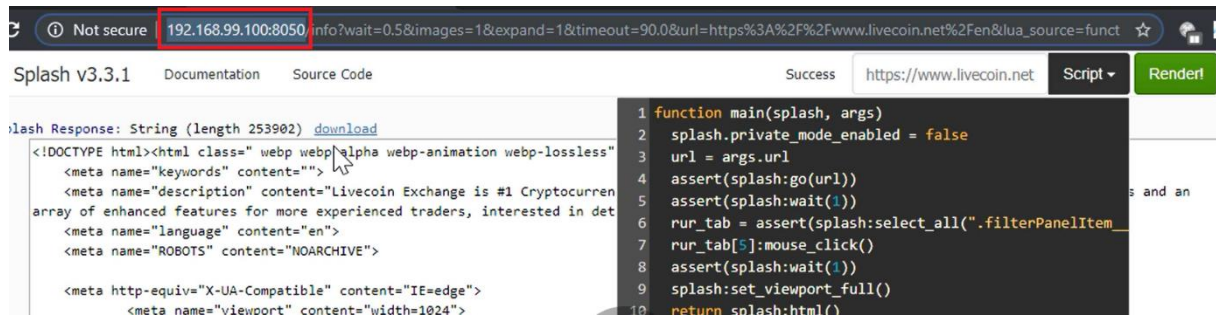
Anaconda prompt'ta oluşturulan projenin içerisindeyken aşağıdaki komutu yapıştırarak indirme yapıyoruz, detaylı adımları videodan da izleyebilirsiniz.

**pip install scrapy-splash**

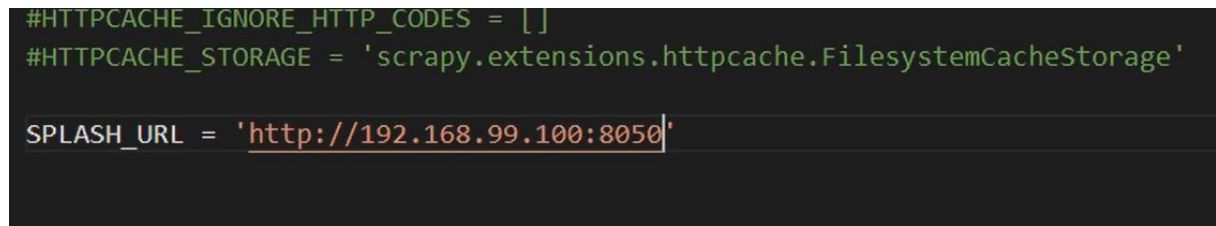
Yüklenen paketi scrapy ile configure edeceğiz.



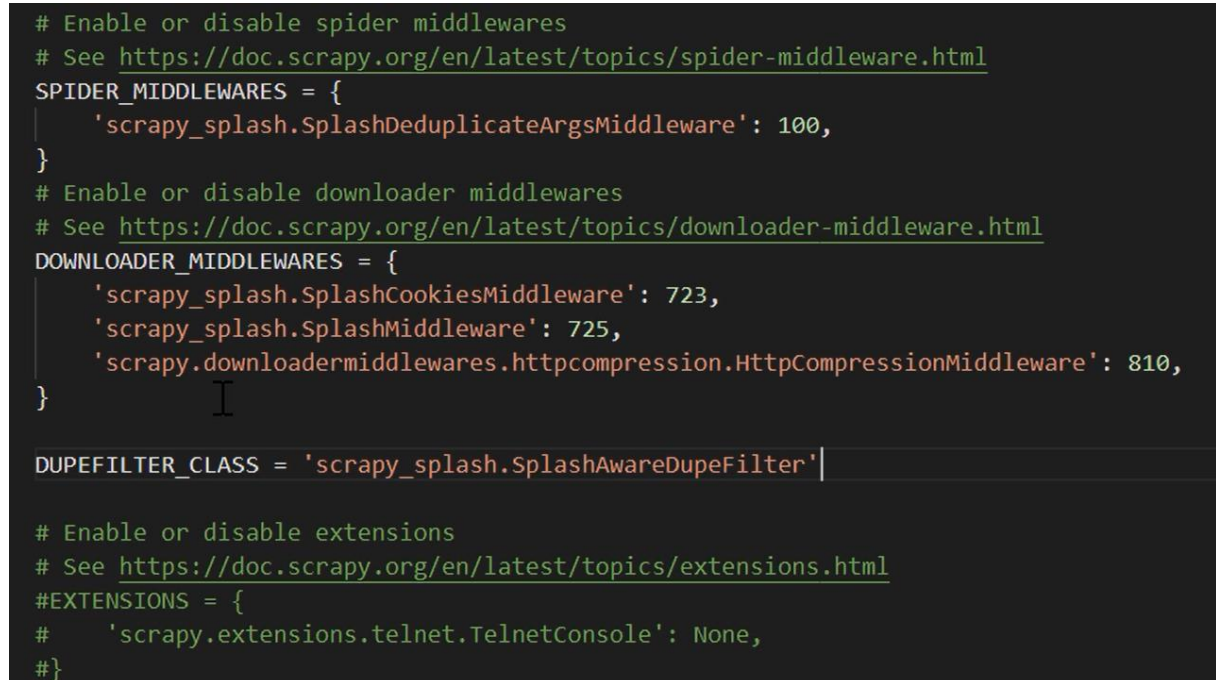
Splash Url'i aşağıdaki splash sayfasından kopyalayıp,



projemin settings.py dosyasının en sonuna yapıştıracağım:



Daha sonra github sayfasından kopyalanan downloader middleware ve spider middleware kodlarını yine settings.py içerisindeki gerekli yerlere yapıştırıyorum, zaten settings.py içerisinde comment içerisinde spidermiddleware ve downloader middleware var oralara yapıştırıyorum.



Son olarak da settings içine yine github'dan kopyaladığımız duefilter satırını yerleştirdik.

## Using the Previous Splash Code inside our Spider

Splash'i scrapy içerisinde kullanabilmek için gerekli olan scrapy-splash paketini kurduğumuza göre şimdi spider'ımıza dönüp, önceki başlıkta livecoin sayfasının USD kısmının markup'ını elde etmek için kullandığımız kodu execute edebiliriz:

Çalıştırmak istediğimiz splash kodu aşağıdaki gibiydi, bunu kopyalıyoruz:

```
1 function main(splash, args)
2   splash.private_mode_enabled = false
3   url = args.url
4   assert(splash:go(url))
5   assert(splash:wait(1))
6   rur_tab = assert(splash:select_all(".filterPanelItem__
7   rur_tab[5]:mouse_click()
8   assert(splash:wait(1))
9   splash:set_viewport_full()
10  return splash:html()
11 end
```

Bu kodu coin spider'ımızı içine aşağıdaki şekilde bir script değişkeni içerisine yerleştiriyoruz:

```
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
```

Splash kullandığımız için start\_request() methodunu override etmek zorundayız, bu yüzden start\_urls listesini siliyoruz.

Splash kullandığımız için scrapy.request() class'ı ile request gönderemeyiz bu çalışmayacaktır.

Bunun için splash'e özel bir başka request class'ını kullanmalıyız. Bunun için önce aşağıdaki class'ı import ediyorum.

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

Daha sonra start\_requests() classını aşağıdaki gibi override ediyorum, ve sanıyorum böylece parse methodu içerisinde splash request'in return'ünü yakalıyoruz. Yani USD sayfasının html markup'ını sanıyorum.

```
assert(splash:go(url))
assert(splash:wait(1))
rur_tab = assert(splash:select_all(".filterPanelItem__2z5Gb"))
rur_tab[5]:mouse_click()
assert(splash:wait(1))
splash:set_viewport_full()
return splash:html()
end

...

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):
    print(response.body)
```

- İlk argüman target website url'i aynı splash UI'sine yazdığımız gibi.
- İkinci argüman callback methodu.
- Üçüncü argüman script içindeki LUA kodu execute etmek istediğimiz için "execute" olarak verildi.
- Son argüman da hangi LUA kodunun execute edileceğini soruyor.

Sonuçta start\_request'i override ederek, bir splash request gönderildi ve splash'in return ettiği değer parse methoduna response olarak verildi ki bu da USD sayfasının html markup'ı oluyor, sanırım normalde de response dediğimiz şey bir html markup oluyor?

## Scraping Elements after Splash Output

Şimdi, splash içerisinde livecoin sayfasında USD butonuna tıkladık ve elde edilen sayfanın html markup'ını return ettirdik, bu markup'a response.body ile ulaşabiliyoruz.

O halde bu markup üzerinde daha önce yaptığımız gibi XPath ile eleman seçimi de yapabiliriz. Bu kısımda işte bunu yapacağız.

Scrape etmek istediğimiz özellikler aşağıda işaretlenen currency ismi ve 24h Volume değeri, diğerleride istenirse alınabilir.

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

İlk önce her satırı seçebileceğimiz bir xpath belirliyoruz, böylece tüm satırlar seçilecek ve daha sonra loop ile her satır içerisinde istenilen elemanlar yield edilebilir.

Bitcoin	BTC/USD	13 484 278.24 USD	9 484.81300 USD	-1.01%	9 632.33719 USD	9 470 USD	
Bitcoin	BCH/USD	155 147.70 USD	248.90000 USD	-1.37%	254.48996 USD	245.00001 USD	
Ethereum	ETH/USD	129 683.59 USD	236.75999 USD	-0.98%	240.37799 USD	233.75743 USD	
Litecoin	LTC/USD	96 822.78 USD	46.15759 USD	-1.11%	47.15023 USD	45.90474 USD	
XEM	XEM/USD	28 372.94 USD	0.04519 USD	-0.7%	0.046 USD	0.045 USD	

Elements

Console

Sources

Network

Memory

Performance

1 of 9

Cancel

```
<div 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;"></div>  
<div aria-rowindex="2" aria-label="row" tabindex="0" class="ReactVirtualized_Table__row tableRow__3Etis odd__YYnX7" role="row" style="height: 50px; left: 0px; position: absolute; top: 50px; width: 1160px; overflow: hidden; padding-right: 0px;"></div>  
<div aria-rowindex="3" aria-label="row" tabindex="0" class="ReactVirtualized_Table__row tableRow__3Etis " role="row" style="height: 50px; left: 0px; position: absolute; top: 100px; width: 1160px; overflow: hidden; padding-right: 0px;"></div>
```

Yukarıdaki xpath expression ile tüm satırlar seçiliyor, her satırın iki tane ortak class'ı olduğu için bu iki class'ı da içeren divleri seçtik.

Şimdi spider içerisinde bu xpath'i response üzerinde kullanacağız:

```
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()
        }
```

Burada yapılan şey şu oldu:

- Splash ile USD sayfasının html markup'ı javascript olmasına rağmen elde edildi.
- Daha sonra ilgili sayfa response olarak parse'a iletildi.
- Aynen daha önce olduğu gibi artık bu response üzerinde xpath ile eleman seçimi yapabilirim.
- Yapılan da bu oldu önce yukarıda belirlenen xpath response üzerinde çalıştırılarak tüm satırlar elde edildi, her bir satır currency'e denk geliyor.
- Daha sonra bir for loop ile her bir satır elemanı yani currency elemanı elde edildi ve daha sonra bu yeni selector elemanı üzerinde tekrar xpath çalıştırılarak istenilen elemanlar yield edildi.
- Yani burada yapılan şey zaten daha önceden bildiğimiz scrapy işleri çok detayına girmiyorum.

Son olarak `scrapy crawl coin -o dataset.csv` ile spider'ımızı çalıştırır ve yielded sonuçları dataset.csv içine kaydedersek, coin datasetimiz hazır olacaktır.