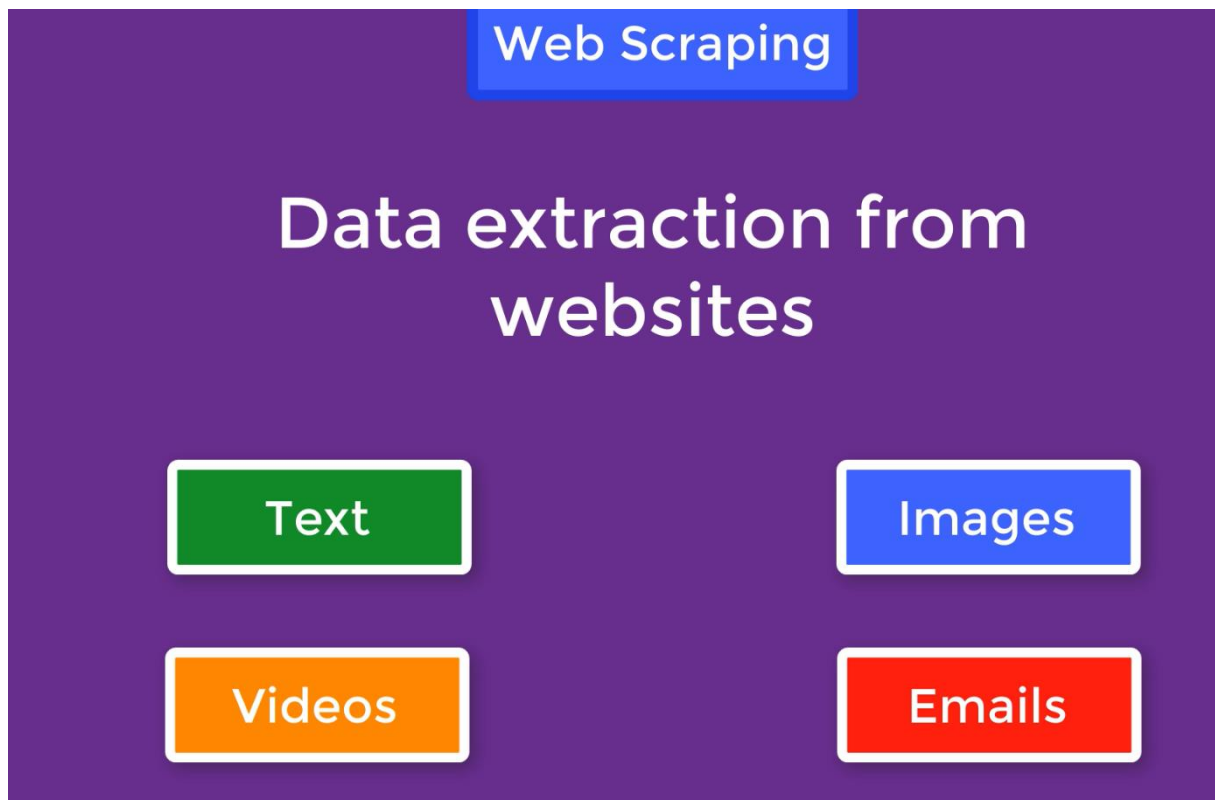


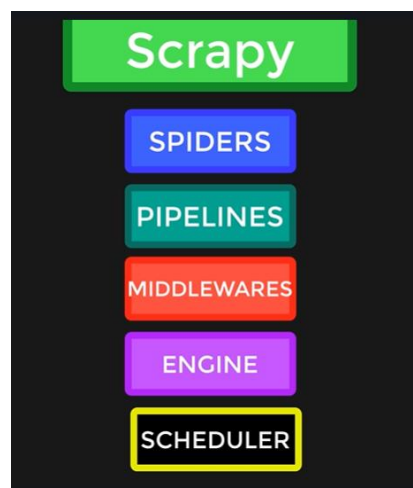
1. INTRODUCTION



Web scrapping için kullanacağımız framework Scrapy olacak. Python'da web scrapping için kullanılabilir başka toollar da var mesela Requests library ile BeautifulSoup'ı kullanarak HTML sayfalardan veri çekebiliriz.

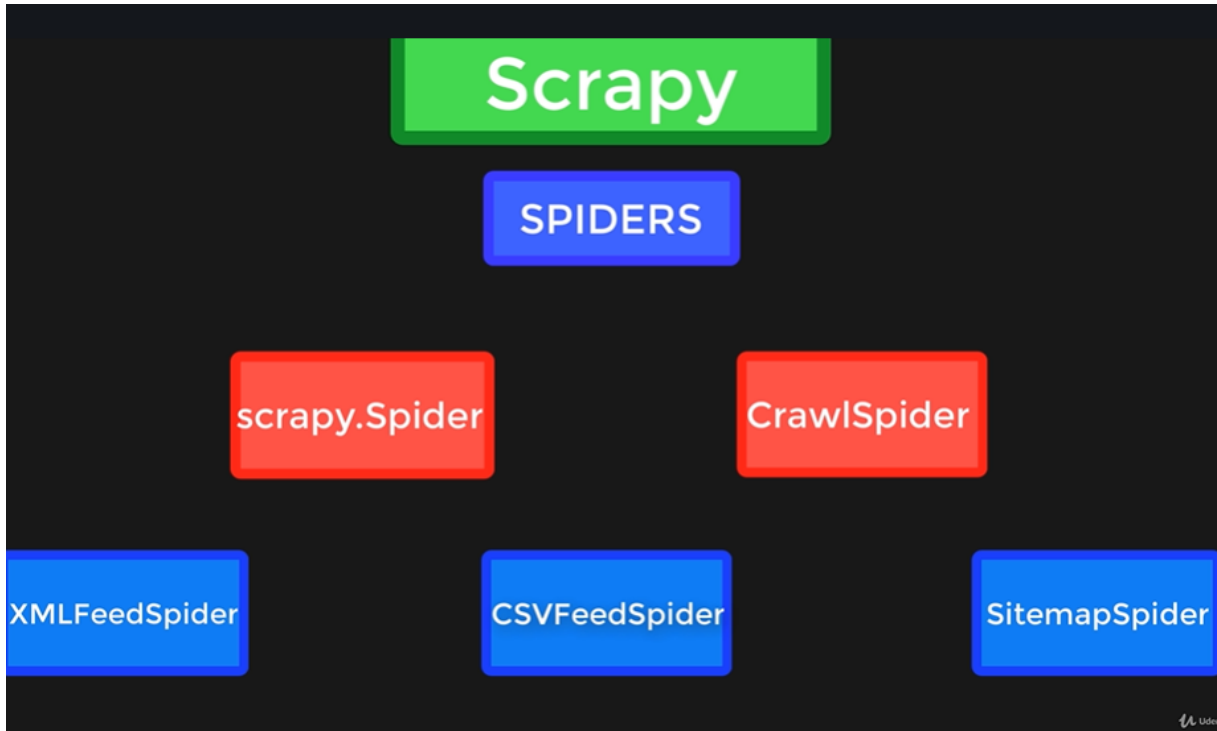
Ancak Requests ve Beautiful Soup ama bu toollar görece basit işler için kullanılır complex projelerde veya çok fazla siteden veri çekme gereksinimi olan durumlarda scrapy'nin avantajını net şekilde görebiliriz.

Şimdi Scrapy architecture'ından bahsedelim, 5 componentten söz edebiliriz:



Şimdi bunları açıklayalım:

- **Spider Component:** Web page'den neyi extract etmek istediğimizi define ettiğimiz yer. Scrapy içinde 5 farklı spider class'ı var:
 - Hepsinin farklı işlevleri var bu kursta sadece kırmızı ile gösterilen iki class'ı kullanacağız.



- **Pipelines Component:** Eğer extract edilen datayı process etmek istiyorsak mesela cleaning, remove duplication etc. veya datayı external bir database'e kaydetmek istiyorsak kesinlikle pipelines kullanmak zorundayız.



- **Middlewares Component:** Website'a request göndermek ve website'ın response'unu almak için gereken component. Örneğin custom headers injection request'ini işlemek istiyorsak ya da proxy kullanmak istiyorsak bu component kullanırız, ilerledikçe bunlar anamlanacak.



- **Engine and Scheduler :** Engine diğer componentler arası koordinasyonu sağlar, scheduler ise order of operation'ı içinde tutar. Teknik olarak scheduler bir queue (FIFO) data structure'ıdır. Bir örnekle anlamaya çalışalım:
 - o Diyelim ki quoteseveryday.com isimindeki siteyi scrape etmek istiyoruz, sitenin içinde bir sürü quotes olduğunu varsayalım biz bunları extract etmek istiyoruz.
 - o Diyelim ki data extract edicek spider'ı build ettik şimdi launching etme zamanı, nasıl çalışacak?

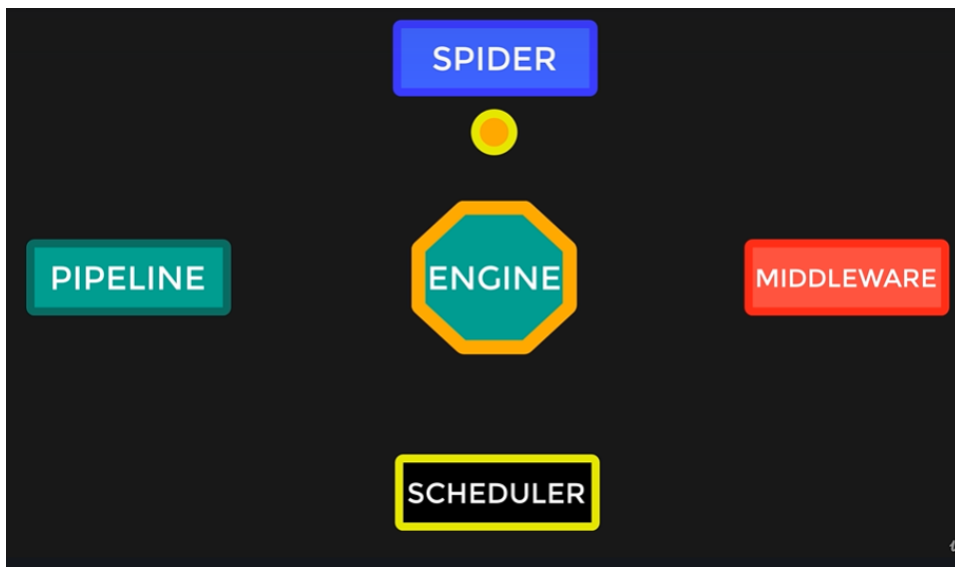
Önce build edilen spider engine'e bir request gönderecek, daha sonra engine bu request'i scheduler'a transmit edecek, scheduler bir FIFO bir queue olduğu için eğer bundan önce bir request gönderilmişse önce onu serve edecek ancak burada başka bir request olmadığını varsayıyoruz, dolayısıyla ilgili request serve edilecek yani engine'e gönderilecek.

Gördüğümüz gibi componentler arası iletişim engine aracılığı ile sağlanıyor.

Daha sonra request engine'den middleware componentine gönderilecek more specifically "Downloader Middleware"e çünkü bu middleware target website'dan response'u almakla yükümlü, middleware'den sonra artık request ile işimiz bitti artık websiteden alınan response elimizde bunu downloader middleware aldı, bu noktadan sonra response işleniyor.

Generated response middleware'den engine'e gönderiliyor, oradan da spider'a gönderiliyor more specifically "Spider Middleware"e. Yani downloader middleware'in işi request'e göre websitesinden response'u almak, spider middleware'in işi ise data extraction'dan , burada data dediğimiz quotes oluyor.

Daha sonra scraped items engine'e gönderilecek oradan da item pipeline'a gönderilecek burada processing yapılacaktır.



Son olarak Robots.txt dosyasından bahsedelim:

Neredeyse bütün websiteleri root directory’de Robots.txt isminde bir dosya içerirler. Bu dosyalar spider’lara access izni olup olmadığı konusunda instructions verirler.

Bu dosya içinde 3 önemle instructions vardır.

- User-Agent: Spider identity’sini temsil eder.
- Allow: Spider’ın hangi sayfalara izni olduğunu belirtir.
- Disallow: Spider’ın hangi sayfalara izni olmadığını belirtir.

Bir örnek olması için facebook.com/robots.txt aşağıdaki gibi görünür

```
https://www.facebook.com/robots.txt
# Notice: Crawling Facebook is prohibited unless you have express written
# permission. See: http://www.facebook.com/apps/site_scraping_tos_terms.php

User-agent: Applebot
Disallow: /ajax/
Disallow: /album.php
Disallow: /checkpoint/
Disallow: /contact_importer/
Disallow: /feeds/
Disallow: /file_download.php
Disallow: /hashtag/
Disallow: /l.php
Disallow: /live/
Disallow: /moments_app/
Disallow: /p.php
Disallow: /photo.php
Disallow: /photos.php
Disallow: /sharer/

User-agent: baiduspider
Disallow: /ajax/
Disallow: /album.php
Disallow: /checkpoint/
Disallow: /contact_importer/
Disallow: /feeds/
Disallow: /file_download.php
Disallow: /hashtag/
Disallow: /l.php
Disallow: /live/
Disallow: /moments_app/
Disallow: /p.php
Disallow: /photo.php
Disallow: /photos.php
Disallow: /sharer/

User-agent: Bingbot
Disallow: /ajax/
Disallow: /album.php
Disallow: /checkpoint/
Disallow: /contact_importer/
Disallow: /feeds/
```

Mesele applebot’un hangi sayfaları izni olmadığını gösteriyor benzer şekilde hangi sayfalara izni olduğu da bu txt içinde yazar.

2. SCRAPY FUNDAMENTALS

PART 1 – Basic Commands

Bu kısımda bazı scrapy command'lerinden ve nasıl kullanılacağından bahsedilecek.

Virtual Environment'in terminaline gelip "scrapy" yazıp enterlarsak available commandsi görebiliriz, bende bu çalışmıyor ama şuan konu işlesin diye devam ediyorum:

```
(virtual_workspace) C:\Users\Ahmed>scrapy
Scrapy 1.6.0 - no active project

Usage:
  scrapy <command> [options] [args]

Available commands:
  bench          Run quick benchmark test
  fetch          Fetch a URL using the Scrapy downloader
  genspider      Generate new spider using pre-defined templates
  runspider      Run a self-contained spider (without creating a project)
  settings       Get settings values
  shell          Interactive scraping console
  startproject   Create new project
  version        Print Scrapy version
  view           Open URL in browser, as seen by Scrapy

[ more ]        More commands available when run from project directory

Use "scrapy <command> -h" to see more info about a command
```

Burada aktif projemiz olmadığını görüyoruz, ayrıca scrapy command usage'ı görebiliyoruz:

- Scrapy <command> [options] [args]

Commands:

- **bench:** benchmark testi çalıştırarak, scrapy'nin hızı ve performansı hakkında bilgi verecektir. "**scrapy bench**" yazarak bu kodu çalıştırabiliriz.
- **fetch:** fetch the HTML markup of specified website. "**scrapy fetch <http://google.com>**" dediğimizde google.com'un raw html markup'ını elde ederiz.
- **genspider:** genelde predefined bir template kullanarak, bir spider generate etmeye yarar. Spider websitelerinden içerik kazımaya yarayan bir component. Google, facebook etc'nin kendi spiderları vardır.

- **runspider**: nadir kullanılır, proje yaratmadan spider'ı çalıştırmaya yarar. Genelde bir proje yaratırız ve proje içinde bir veya daha fazla spider yaratırız. Ancak bazen hızlı olması için proje yaratmadan doğrudan spider'ı yaratıp test etmek isteyebiliriz.
- **settings**: default settings'i görmeye yarar.
- **shell**: önemli bir command, bunu genelde scrape edilecek website üzerinde spider yaratmadan önce bazı denemeler yapmak için kullanılır
- **startproject**: projeye başlamak için gerekli dosyaları yaratır.
- **version**: versiyonu yazdırır.
- **view**: browser'ı kullanarak specified website'ı açar ve scrapy tarafından sitenin nasıl görüldüğünü gösterir. Ancak buna güvenmiyormuş, bunun alternatifini daha sonra gösterecekmış.
- **[more]**

PART 2 – Create a Project and a Spider

Bir scrapy projesi yaratarak işe koyulalım, öncelikle environment terminalde cd ile istediğimiz klasöre geliyoruz ben masaüstünde bir projeler klasörü yarattım.

Current directory'ı bu klasör olarak atadıktan sonra **scrapy startproject worldometers** diyerek bir proje oluşturuyoruz burada bir template kullanarak scrapy projesinin oluşturulduğunu söylüyor.

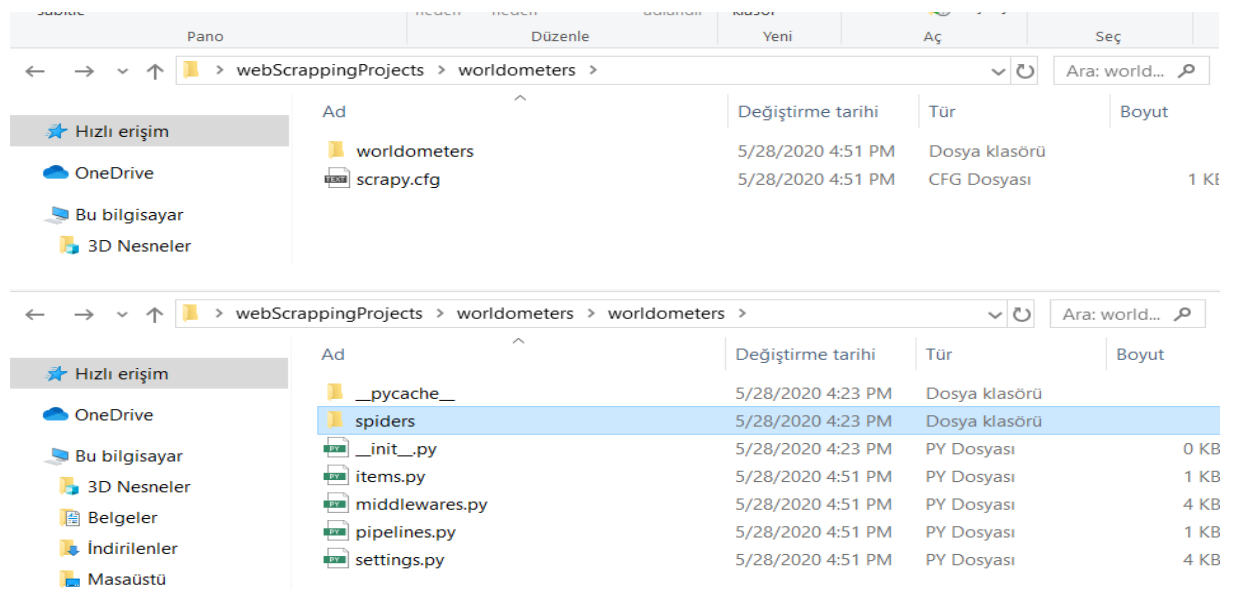
Daha sonra **cd worldometers** ve **scrapy genspider example example.com** komutlarıyla spider'ımızı generate edebileceğimizi söylüyor.

```
(virt_env) C:\Users\Seko>cd Desktop
(virt_env) C:\Users\Seko\Desktop>mkdir webScrapingProjects
(virt_env) C:\Users\Seko\Desktop>cd webScrapingProjects
(virt_env) C:\Users\Seko\Desktop\webScrapingProjects>scrapy startproject worldometers
New Scrapy project 'worldometers', using template directory 'E:\Anaconda\Anaconda3\lib\site-packages\scrapy\templates\project', created in:
  C:\Users\Seko\Desktop\webScrapingProjects\worldometers
You can start your first spider with:
  cd worldometers
  scrapy genspider example example.com
(virt_env) C:\Users\Seko\Desktop\webScrapingProjects>
```

Masaüstünde yaratılan dosyalara bakarsak, Project klasörünün içinde bir worldometers proje klasörü yer alıyor, onun içinde bir config file ile bir başka worldometers file'ı yer alıyor.

Buradaki config file yaratılan spider'ın execute edilmesi için önem arz ediyor ayrıca spider'ımızı Heroku, ScrapingHub cloud, Scrapy daemon gibi platformlarda deploy etmemiz için önemli bunları kursun sonlarında göreceğiz.

Worldometers klasörünün içinde ise birçok başka file var. Örneğin bir spider klasöründe, yaratılacak spiderlar tutulur, şuanda burada boş bir init file yer alır.



Items.py dosyasını scrape edilen datayı temizlemek için veya fields içinde store etmek için kullanılır. Field'ler aşağıdaki gibi tanımlanabilir. Bunları görecez.

```
items.py
1 # -*- coding: utf-8 -*-
2
3 # Define here the models for your scraped items
4 #
5 # See documentation in:
6 # https://doc.scrapy.org/en/latest/topics/items.html
7
8 import scrapy
9
10
11 class WorldometersItem(scrapy.Item):
12     # define the fields for your item here like:
13     # name = scrapy.Field()
14     pass
15
```


Benzer şekilde `middlewares.py` dosyası var, bu dosyanın olayı request ve response'ları handle etmesi. 2 çeşit middlewares olduğunu görmüştük:

- Spider middleware: responsible for returning back the data.
- Downloader middleware: responsible for the downloading the HTML markup of a website.

Middlewares.py dosyasının içinde iki middleware class'ı da yer alır. Daha sonra fazlasını göreceğiz.

```
1 class WorldometersSpiderMiddleware(object):
2     # Not all methods need to be defined. If a method is not defined,
3     # scrapy acts as if the spider middleware does not modify the
4     # passed objects.
5
6     @classmethod
7     def from_crawler(cls, crawler):
8         # This method is used by Scrapy to create your spiders.
9         s = cls()
10        crawler.signals.connect(s.spider_opened, signal=signals.spider_opened)
11        return s
12
13    def process_spider_input(self, response, spider):
14        # Called for each response that goes through the spider
15        # middleware and into the spider.
16
17        # Should return None or raise an exception.
```

```
class WorldometersDownloaderMiddleware(object):
    # Not all methods need to be defined. If a method is not defined,
    # scrapy acts as if the downloader middleware does not modify the
    # passed objects.

    @classmethod
    def from_crawler(cls, crawler):
        # This method is used by Scrapy to create your spiders.
        s = cls()
        crawler.signals.connect(s.spider_opened, signal=signals.spider_opened)
        return s

    def process_request(self, request, spider):
        # Called for each request that goes through the downloader
        # middleware.
```

`pipelines.py` ise scrape edilen itemları database içinde store etmek için kullanılır.

Son olarak `settings.py` projeye extra configuration eklemek gerektiğinde kullanılır, bu file'ı sıklıkla kullanacağız.

Şimdi worldometers projemize dönüp bir spider yaratacağız ancak öncelikle scrape edeceğimiz websitesine bir göz gezdirelim:

<https://www.worldometers.info/world-population/population-by-country/>

Burada ülkelere tıklayarak yeni bir sayfada hangi yıllarda ilgili ülkenin nüfusu kaçtı görebiliriz.

Şimdiye kadar aşağıdaki şekilde scrapy projemizi yaratmıştık, spider'ı nasıl yaratacağımızı da gösteriyor:

```
(virt_env) C:\Users\Seko>cd Desktop
(virt_env) C:\Users\Seko\Desktop>mkdir webScrappingProjects
(virt_env) C:\Users\Seko\Desktop>cd webScrappingProjects
(virt_env) C:\Users\Seko\Desktop\webScrappingProjects>scrapy startproject worldometers
New Scrapy project 'worldometers', using template directory 'E:\Anaconda\Anaconda3\lib\site-packages\scrapy\templates\project', created in:
  C:\Users\Seko\Desktop\webScrappingProjects\worldometers
You can start your first spider with:
  cd worldometers
  scrapy genspider example example.com
(virt_env) C:\Users\Seko\Desktop\webScrappingProjects>
```

Şimdi belirtildiği gibi **cd worldometers** dedikten sonra **scrapy genspider countries www.worldometers.info...** şeklinde spider klasörü içinde **countries.py** isimindeki spider'ımı yaratabiliyorum. Web sitesi ismini / olmadan girmeye dikkat et.

```
(virt_env) C:\Users\Seko\Desktop\webScrappingProjects>cd worldometers
(virt_env) C:\Users\Seko\Desktop\webScrappingProjects\worldometers>scrapy genspider countries www.worldometers.info/world-population/population-by-country
Created spider 'countries' using template 'basic' in module:
  worldometers.spiders.countries
(virt_env) C:\Users\Seko\Desktop\webScrappingProjects\worldometers>
```

Basic template'ı kullanılarak countries spider'ı oluşturuldu diyor:

webScrappingProjects > worldometers > worldometers > spiders			
Ad	Değiştirme tarihi	Tür	
__pycache__	5/28/2020 6:16 PM	Dosya klasörü	
__init__.py	5/28/2020 4:23 PM	PY Dosyası	
countries.py	5/28/2020 6:16 PM	PY Dosyası	

Şimdi countries.py spider'ını açalım ve bir spider'ın nasıl görüldüğüne bakalım:

```
1 # -*- coding: utf-8 -*-
2 import scrapy
3
4
5 class CountriesSpider(scrapy.Spider):
6     name = 'countries'
7     allowed_domains = ['www.worldometers.info/world-population/population-by-country']
8     start_urls = ['http://www.worldometers.info/world-population/population-by-country/']
9
10    def parse(self, response):
11        pass
12
```

Temelde CountriesSpider'ı scrapy.Spider classının bir subclass'ından fazlası değil.

Her spider'ın bir ismi vardır, bunun ismine countries dedik. Bir projede birden fazla spider olabilir hepsinin unique bir ismi olmalı.

Allowed_domains içerisinde spider'ın scrape etmeye izni olan domain'ler yer alır mesela sitede bir link varsa bu başka bir sayfaya gidiyorsa ve o sayfa bu listede yer almıyorsa, spider o sayfada scraping yapamaz.

Ayrıca bu domain ismi asla http:// ile başlamamalı, yoksa hata verir.

start_urls: buraya scrape etmek istediğimiz tüm linkleri yazıyorum bizim scrape edeceğimiz websitesi linki https:// protokolü ile başlıyor bunu değiştiriyoruz.

Bu CountriesSpider subclassının içine yazılanlar aslında Spider class'ının bazı attribute'larının ve parse metodunun'ının override edilmiş halinden fazlası değil, isimler tam olarak böyle olmalı. CountriesSpider'ımız bir spider'ın taşıdığı bütün özellikleri taşıyor, sadece bazı özelliklerini override ettik.

Son olarak spider içerisinde bir parse metodu var, burada spider'dan alınan response'u parse ederiz. Yani start_urls ile belirtilen linke request gönderildiğinde response'u bu parse method ile yakalayacağız

PART 3 - Shell

Bu kısımda genelde spider'ımızı build etmeden önce kullandığımız bir tool olan scrapy shell'i öğreneceğiz, basic element selection ve debugging XPath expressions or CSS selectors için kullanılır.

Öncelikle conda install ipython ile environment'ımıza ipython'ı yükledik, şimdi **bir scrapy shell launch edelim:**

Environment terminal'e

scrapy shell

commandini giriyoruz, istersek url'i de yanına "url" şeklinde girebiliriz, ancak bunu shell'i oluşturduktan sonra da girebiliriz, öyle yapacağız:

Şimdi bazı available scrapy objects'e erişimimiz olduğunu ve bazı shortcutları aşağıda görüyoruz:

```
[s] Available Scrapy objects:
[s] scrapy      scrapy module (contains scrapy.Request, scrapy.Selector, etc)
[s] crawler     <scrapy.crawler.Crawler object at 0x0000027893983470>
[s] item        {}
[s] settings    <scrapy.settings.Settings object at 0x0000027896084940>
[s] Useful shortcuts:
[s] fetch(url[, redirect=True]) Fetch URL and update local objects (by default, redirects are followed)
[s] fetch(req)                  Fetch a scrapy.Request and update local objects
[s] shelp()                     Shell help (print this help)
[s] view(response)             View response in a browser
In [1]:
```

Url'i açmak için fetch methodunu kullanıyoruz url'i argument olarak bu methodun içine pass edebiliriz veya 2. Kullanımdaki gibi önce bir request object oluştururuz ve fetch'in içine bu request object'i pass ederiz. İki yöntemi de göreceğiz

Response'u browser'da görmek için de view komutu kullanılabilir ancak bunu çok sık kullanmadığını söylüyor, yine de istenilen webpage'i mi yoksa captcha gibi bir sayfayı mı response olarak aldığını görmek için kullanılabilir.

Şimdi fetch komutuyla url'yi fetch edelim.

```
fetch("https://www.worldometers.info/world-population/population-by-country/")
```

```
In [4]: fetch("https://www.worldometers.info/world-population/population-by-country")
2020-05-29 12:53:18 [scrapy.core.engine] INFO: Spider opened
2020-05-29 12:53:19 [scrapy.downloadermiddlewares.redirect] DEBUG: Redirecting (301) to <GET https://www.worldometers.info/world-population/population-by-country/> from <GET https://www.worldometers.info/world-population/population-by-country/>
2020-05-29 12:53:19 [scrapy.core.engine] DEBUG: Crawled (200) <GET https://www.worldometers.info/world-population/population-by-country/> (referer: None)

In [5]:
```

- Spider opened diyor.
- GET gönderilen request tipi gibi düşünülebilir. Bunun yanında ayrıca POST request'i de vardır bu da websitesine bilgi gönderirken kullanılır mesela form doldururken vesaire.
- Crawled (200) <GET ... satırı bize şunu diyor, ilgili URL request'ini yerine getirdik ve karşılığında bir response elde ettik.

Scrapy shell ile bir URL'i açmanın bir diğer yolu ise önce bir request objesi oluşturup bu objeyi fetch içine beslemektir şimdi de bunu yapalım:

```
r = scrapy.Request(url="https://www.worldometers.info/world-population/population-by-country/")
```

```
fetch(r)
```

```
In [5]: r = scrapy.Request(url="https://www.worldometers.info/world-population/population-by-country/")

In [6]: fetch(r)
2020-05-29 13:00:55 [scrapy.core.engine] DEBUG: Crawled (200) <GET https://www.worldometers.info/world-population/population-by-country/> (referer: None)

In [7]:
```

- scrapy.Request() içine url ve başka args alan bir scrapy classı.
- Gördüğümüz gibi biraz öncekiyle aynı fetch sonucunu elde ediyoruz.

Şimdi **response.body** diyerek fetch edilen sayfanın html markup'ını output edebiliriz. Burada elde edilen html markup sayfaya gidip **CTRL+U** yaptığımızda göreceğimiz markup ile aynı.

Son olarak **view(response)** komutu ile response'u browser üzerinde görebiliriz. Burada açılan sayfa orijinali gibi görünür, ancak aslında spider sayfayı tam olarak böyle görmez, spider sayfayı JavaScript olmadan görür, yani javascript ile yaratılan şeylerin hiçbirini göremez.

Bu görünüşü anlamak için sayfaya gidip developer toolu açarız CTRL+SHIFT+P ile command palette'i açıp search kısmına javascript yazarak disable javascripti seçebiliriz ve CTRL+R ile sayfayı yenileyince sayfa javascript engellenmiş olarak görünecek.

Sonuçta scrapy spider'ları javascript'i render edemiyor bu bilgi önemli bunu gözden geçirirsek hatalara neden olabilir.

PART 4 - Selectors

Bu kısımda XPath expressions ve CSS selectors ile scrapy’de nasıl element selection yapılacağını göreceğiz. XPath expressions ve CSS selectors bir sonraki kısımda detaylıca anlatılacak, şuan dert etme anlamaya çalış.

Şimdi diyelim ki XPath kullanarak URL’imizdeki title’ı scrape etmek istiyoruz:

[W](#) / [Population](#) / Population by Country

Countries in the world by population (2020)

This list includes both **countries** and **dependent territories**. Data based on the latest *United Nations Population Division* estimates.

Click on the name of the country or dependency for current estimates (live population clock), historical data, and projected figures.

See also: [World Population](#)

#	Country (or dependency)	Population (2020)	Yearly Change	Net Change	Density (P/Km ²)	Land Area (Km ²)	Migrants (net)	Fert. Rate	M A
---	-------------------------	-------------------	---------------	------------	------------------------------	------------------------------	----------------	------------	-----

Hata yapmamak için öncelikle kendi web browser’ımızdan Javascript’i disable ediyoruz, çünkü selectors’ları önce kendimiz browser’dan sayfa markup’ına bakıp öyle oluşturacağız javascript’i kapatmazsak sıkıntı çıkabilir. Kapatmak için F12, ardından CTRL+Shift+P ile paleti açarız ve javascript yazıp disable’i seçtikten sonra sayfayı refresh ederiz (CTRL+R)

Bu işlemten sonra sayfadan scrape etmek istediğimiz title'ı inspect edip markup'ına bakıyoruz:

The screenshot shows a web browser window with the title 'Population by Country' and a page with the heading 'Countries in the world by population (2020)'. Below the heading is a paragraph of text and a table. The Chrome DevTools 'Elements' panel is open, showing the DOM tree. The selected element is an

tag with the text 'Countries in the world by population (2020)'. The breadcrumb path is: > <div class="col-md-12"> > <h1>Countries in the world by population (2020)</h1> == \$0.

Görüyoruz ki bir div içerisindeki bir h1 etiketinin içine title gömülmüş. Şimdi yine browser üzerinde XPath expression'inimizi yazacağız ve gerçekten elde etmek istediğimiz title'ı elde edebiliyor muyuz ona bakacağız.

Inspector window'da elements tab'ı seçiliyken CTRL+F ile find by string, selector or Xpath seçeneğini açarız. Burada //h1 ile Xpath tanımlarız ve otomatik olarak bulunan elementler işaretlenir bu sayfada tek bir h1 etiketi olduğu için sadece başlığı buldu.

The screenshot shows the Chrome DevTools 'Elements' panel with the XPath expression '//h1' entered in the search bar. The search results show a single element: <h1>Countries in the world by population (2020)</h1> == \$0. The breadcrumb path is: html > body > div.container > div.row > div.col-md-12 > div.content-inner > div.col-md-12 > h1. The search bar shows the XPath expression '//h1' and the results show 1 of 1 matches.

Artık şunu anladıkki bu sayfada `//h1` Xpath'ini kullanarak başlığa ulaşabiliyoruz, şimdi daha önce elde edilen response'u kullanarak bu başlığa terminalden ulaşacağız:

Üçüncü kısımda url'yi fetch etmiş ve sayfadan bir response elde etmiştik şimdi bu response'u kullanarak istediğimiz başlığa ulaşalım:

`title = response.xpath("//h1")` response içinden h1 etiketini bul dedik.

Şimdi title dediğimizde bir listenin içinde 2 property'si olan bir Selector object ile karşılaşırız:

```
In [10]: title = response.xpath("//h1")
In [11]: title
Out[11]: [<Selector xpath='//h1' data='<h1>Countries in the world by populat...>']
In [12]:
```

List içindeki selector objesinin ilk property'si bizim girdiğimiz xpath'l gösteriyor ikinci property'si de ilgili xpath ile response'dan alınan datayı gösteriyor ki bu tek bir h1 etiketine karşılık geliyor.

Eğer datayı html markup element değil de içerisindeki text content olarak elde etmek istiyorsak yapacağımız şey şu:

`title = response.xpath("//h1/text()")`

```
In [13]: title = response.xpath("//h1/text()")
In [14]: title
Out[14]: [<Selector xpath='//h1/text()' data='Countries in the world by population ...>']
In [15]:
```

Gördüğümüz gibi bu kez selector'ın data kısmı text olarak elde edildi.

Eğer bu datayı string olarak return etmek istersek:

`title.get()`

```
In [15]: title.get()
Out[15]: 'Countries in the world by population (2020)'
In [16]:
```

Gördüğümüz gibi title'l elde etmiş olduk.

Şimdi aynı title scrape işlemini XPath yerine Css selectors kullanarak yapalım:

title = response.xpath("//h1") yerine

title_css = response.css("h1::text") kullanırız, şunu demiş oluyoruz, response'un içinde css selector ile h1 etiketinin içindeki text'i al title_css içine kaydet.

Title_css'ı yazdırdığımız zaman yine bir list içerisinde Selector object ile karşılaşırız, yine bir xpath'ı ve data'sı vardır, yani yazdığımız css selector aslında bir xpath'e çevrildi o yüzden css selector kullanmak pek verimli değil, sonuçta aynı şekilde h1 etiketini data içinde elde etmiş olduk.

```
In [16]: title_css = response.css("h1::text")
In [17]: title_css
Out[17]: [<Selector xpath='descendant-or-self::h1/text()' data='Countries in the world by population ...'>]
In [18]: title_css.get()
Out[18]: 'Countries in the world by population (2020)'
In [19]:
```

Az önceki gibi elde ettiğimiz title_css içerisinden get() method ile datayı çekebiliyoruz.

Şimdi diyelim ki başlığı değil de sayfadaki tüm ülkeleri çekmek istiyorum (Xpath ile yapalım):

İlk ülke elemanına bakıyorum bir td elemanı içinde yer alan a elemanı içerisinde country name text olarak tutuluyor.

#	Country (or dependency)	Population (2020)	Yearly Change	Net Change	Density (P/Km²)	Land Area (Km²)	Migrants (net)	Fert. Rate	M A.
1	China	1,439,323,776	0.39 %	5,540,090	153	9,388,211	-348,399	1.7	36
2	India	1,380,004,385	0.99 %	13,586,631	464	2,973,190	-532,687	2.2	26
3	United States	331,002,651	0.59 %	1,937,734	36	9,147,420	954,806	1.8	36
4	Indonesia	273,523,615	1.07 %	2,898,047	151	1,811,570	-98,955	2.3	36
5	Pakistan	220,892,340	2.00 %	4,327,022	287	770,880	-233,379	3.6	26
6	Brazil	212,559,417	0.72 %	1,509,890	25	8,358,140	21,200	1.7	36

https://www.worldometers.info/world-population/china-population/

Elements Console Sources Network Memory Performance >> ⚙

<tbody>

<tr>

<td>1</td>

<td style="font-weight: bold; font-size:15px; text-align:left">

China == \$0

</td>

<td style="font-weight: bold;">1,439,323,776</td>

<td>0.39 %</td>

<td>5,540,090</td>

<td>153</td>

<td>9,388,211</td>

<td>-348,399</td>

<td>1.7</td>

<td>36</td>

O zaman önce CTRL+F ile Xpath expression olarak //a vermeyi deneyelim, ancak göreceğiz ki bu yetersiz kalacak çünkü başka a elemanları da bulunacak sadece ülkeler değil onun için Xpath olarak td içerisindeki a elemanlarını belirteceğiz:

//td/a

#	Country (or dependency)	Population (2020)	Yearly Change	Net Change	Density (P/Km²)	Land Area (Km²)	Migrants (net)	Fert. Rate	M A
1	China	1,439,323,776	0.39 %	5,540,090	153	9,388,211	-348,399	1.7	38
2	India	1,380,004,385	0.99 %	13,586,631	464	2,973,190	-532,687	2.2	28
3	United States	331,002,651	0.59 %	1,937,734	36	9,147,420	954,806	1.8	38
4	Indonesia	273,523,615	1.07 %	2,898,047	151	1,811,570	-98,955	2.3	30
5	Pakistan	220,892,340	2.00 %	4,327,022	287	770,880	-233,379	3.6	23

Tamam browser üzerinden Xpath expression'ın çalıştığını anlamış olduk. Şimdi Shell üzerinden yaptığımız fetch komutuyla elde edilen url response'una geri dönelim ve o response üzerinden xpath tanımlayıp ülke adlarını elde edelim:

```
countries = response.xpath("//td/a/text()")
```

```
In [21]: countries = response.xpath("//td/a/text()")
In [22]: countries
Out[22]:
[<Selector xpath='//td/a/text()' data='China'>,
 <Selector xpath='//td/a/text()' data='India'>,
 <Selector xpath='//td/a/text()' data='United States'>,
 <Selector xpath='//td/a/text()' data='Indonesia'>,
 <Selector xpath='//td/a/text()' data='Pakistan'>,
```

Sonuçta bu sefer belirtilen xpath ile 233 tane element olduğu için countries listesinin içine her bir elementi bir selector object olarak kaydettik eğer

countries.get() dersek sadece ilk selector'ın data'sını yani China'yı elde ederiz.

countries.getall() dersek tüm selector'ların datalarını bir liste olarak return ederiz.

Tüm ülkeleri çekmek istersem (CSS Selector ile):

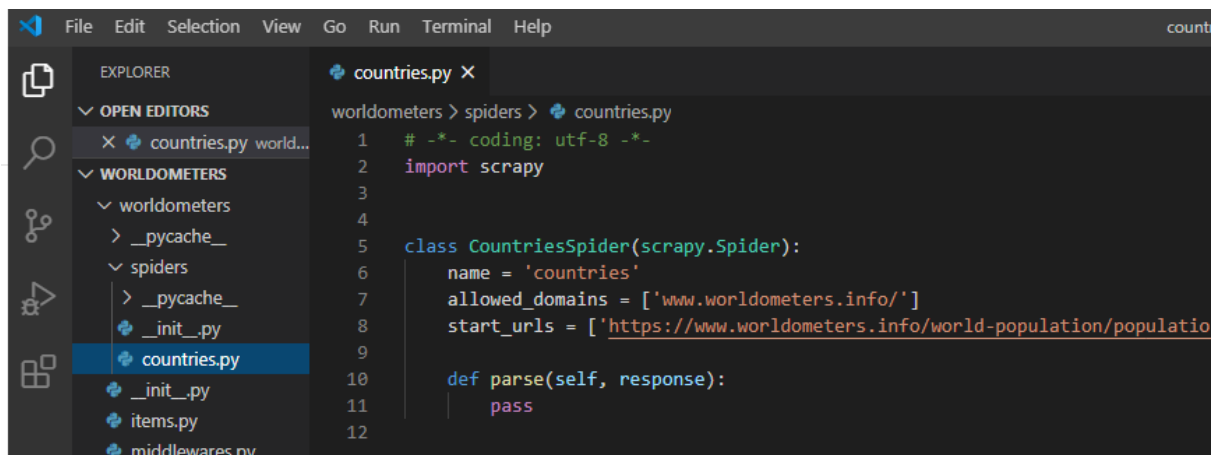
```
countries = response.css("td a::text").getall()
```

PART 5 – Building our first Spider

Part 2 de bir proje ve içinde bir spider generate etmemize ragmen henüz bu projeyi hiç kullanmadık, part 3 ve part 4'te yapılanlar scrapy shell üzerinden yapıldı, shell üzerinden url'yi fetch ettik ve siteye bir request gönderdik ve response'u alıp içerisinden çeşitli etiketleri çekmeyi başardık, bunların hepsi test amaçlı yapılan şeylerdi.

Şimdi bu request işlemini ve gelen response'u handle etme işini oluşturduğumuz spider class'ı içerisinden yapacağız.

VSCode editöründen `file>open>projectKlasörü>worldometer` ile proje klasörümüzü seçip açıyoruz, buradan da spider klasörü içerisinde daha önce generate edilen `countries.py` spider'ına giriyoruz:

The screenshot shows the VS Code editor interface. On the left, the Explorer sidebar is open, showing a project structure with folders like 'worldometers' and 'spiders'. The 'countries.py' file is selected and open in the editor. The code in the editor is as follows:

```
1  # -*- coding: utf-8 -*-
2  import scrapy
3
4
5  class CountriesSpider(scrapy.Spider):
6      name = 'countries'
7      allowed_domains = ['www.worldometers.info/']
8      start_urls = ['https://www.worldometers.info/world-population/population-by-country/']
9
10     def parse(self, response):
11         pass
12
```

Bu spider'ı generate edersek zaten url'yi girmiştik, yukarıdada `start_urls` kısmındaki url spider'ın request göndereceği ve karşılığında response alacağı url.

Alınan response ile yapılacak işlemler ise parse methodunun içinde yapılacak, daha önce shell üzerinde yaptığımız işlemleri burada yapalım:

Parse methodu içinde response'dan hem `title`'ı hem de `countries`'i çekiyorum ve bir dictionary olarak yield ediyorum.

```

class CountriesSpider(scrapy.Spider):
    name = 'countries'
    allowed_domains = ['www.worldometers.info/']
    start_urls = ['https://www.worldometers.info/world-population/population-by-country/']

    def parse(self, response):
        #response üzerinden h1 etiketinin içindeki text'i çekelim:
        #burada tek selector objesi döndürülecek çünkü tek bir h1 etiketi var.
        #selector içinde xpath ve çekilen etiket datası yer alacak.
        title = response.xpath("//h1/text()").get()

        #response üzerinde td/a etiketlerinin içindeki text'i list olarak çekelim:
        #burada 233 tane selector döndürülecek.
        #çünkü 233 tane ilgili etiketten var.
        #her selector'ın datasını list olarak çekmek için getall() kullandık.
        countries = response.xpath("//td/a/text()").getall()

        #Scrapy'de return edilecek datayı her zaman bir dict olarak return etmeliyiz:
        #yield'i return gibi düşün, detayın istersen bak.
        yield{
            'title':title
            'countries':countries
        }

```

Spider'ımız hazır, dosyayı kaydettikten sonra anaconda prompt üzerinden spider'ı execute edebiliriz.

Bunun için önce scrapy.cfg file'ı ile aynı level'a gelmeliyiz, bu levelda olup olmadığımızı dosya path'ine bakarak check edebileceğimiz gibi, terminaldeyken **dir** komutunu kullanarak da anlayabiliriz, bu komut ile current directory içerisindeki dosyaları görürüz, eğer doğru cd'de isek scrapy.cfg görünmeli:

```

(virt_env) C:\Users\Seko>cd Desktop/webScrapingProjects/worldometers

(virt_env) C:\Users\Seko\Desktop\webScrapingProjects\worldometers>dir
Volume in drive C has no label.
Volume Serial Number is 48D8-100D

Directory of C:\Users\Seko\Desktop\webScrapingProjects\worldometers

05/28/2020  04:51 PM    <DIR>          .
05/28/2020  04:51 PM    <DIR>          ..
05/28/2020  04:51 PM                267 scrapy.cfg
05/28/2020  04:51 PM    <DIR>          worldometers
               1 File(s)                267 bytes
               3 Dir(s)  21,823,561,728 bytes free

(virt_env) C:\Users\Seko\Desktop\webScrapingProjects\worldometers>

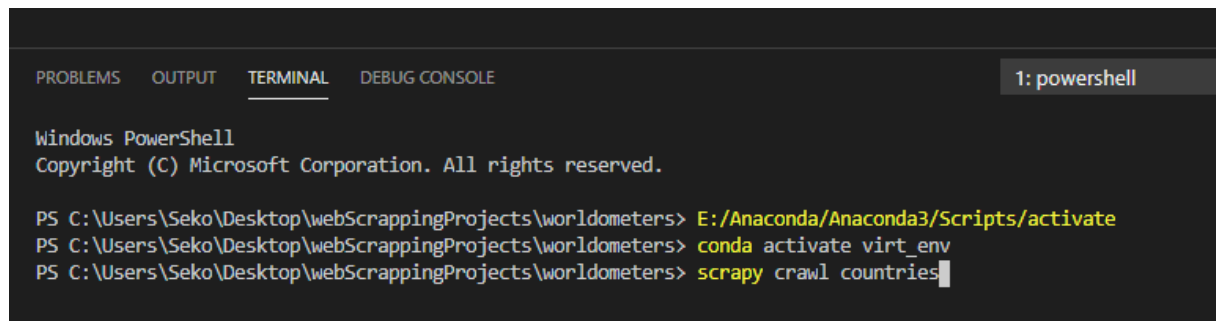
```

Doğru current directory’de olduğumuzdan emin olduktan sonra spider’ı execute etmek için şu komutu kullanıyoruz:

`scrapy crawl spiderName` yani bizim durumumuzda spider ismi countries idi o halde:

`scrapy crawl countries`

Aynı komutu vs code editörü üzerinde çalıştırmak için `View>Terminal` diyoruz ve aşağıda açılan terminale aynı komutu yazıp çalıştırıyoruz:



```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE  1: powershell

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Users\Seko\Desktop\webScrappingProjects\worldometers> E:/Anaconda/Anaconda3/Scripts/activate
PS C:\Users\Seko\Desktop\webScrappingProjects\worldometers> conda activate virt_env
PS C:\Users\Seko\Desktop\webScrappingProjects\worldometers> scrapy crawl countries
```

Bu şekilde spider’ımızı çalıştırmış olduk ve websitesinden header ile country isimlerini title ve countries değişkenlerinin içine kaydettik.

3.XPath Expressions & CSS Selectors

2. Başlıkta Scrapy Fundamentals'ı gördük, shell tool'u ile seçilen websitesi için request gönderdik ve alınan response üzerinden title ve countries'e ulaştık daha sonra aynı işlemi spider class'ı içinde tanımladık parse metodu ile response'ı handle ettik.

Bu başlık altında Xpath expression ve CSS Selectors hakkındaki bilgimizi genişleteceğiz çünkü bu konu web scrapping için çok önemli.

Önceki kısımda sadece basit element selections yaptık, elbette bu konuda öğrenecek daha fazla şey var bu başlık altında bunları görelim:

Understanding XPath & CSS Selectors

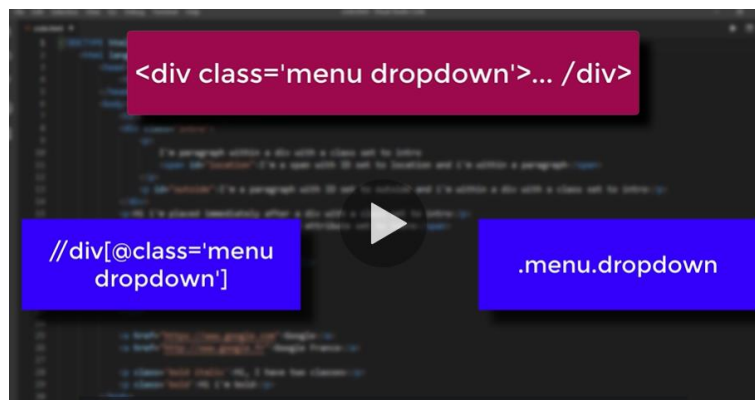
XPath'in ve CSS Selectors'ın ne olduğunu anlamaya çalışalım.

XPath : XML Path Language anlamına gelir, yani it's a language that was mainly created to query XML documents but it doesn't mean that we can't query or select elements or tags from HTML webpages.

CSS: Cascading Style Sheet, language mainly created to style HTML pages.

Xpath'in de CSS Selectors methodunun da iyi olduğu durumlar vardır, bunlardan bahsedilecek.

Ancak özetle Xpath'in avantajı, HTML webpage üzerinde yukarı ve aşağı gitmemize olanak vermesidir, CSS ile bunu yapamayız. Buna karşın banze CSS syntax'i Xpath'den daha iyi görünür.



Yukarıdaki gibi bir class div'ı seçmek için soldaki Xpath syntax veya sağdaki CSS syntax kullanılacak, sağdaki daha temiz görünüyor.

CSS Selectors Fundamentals

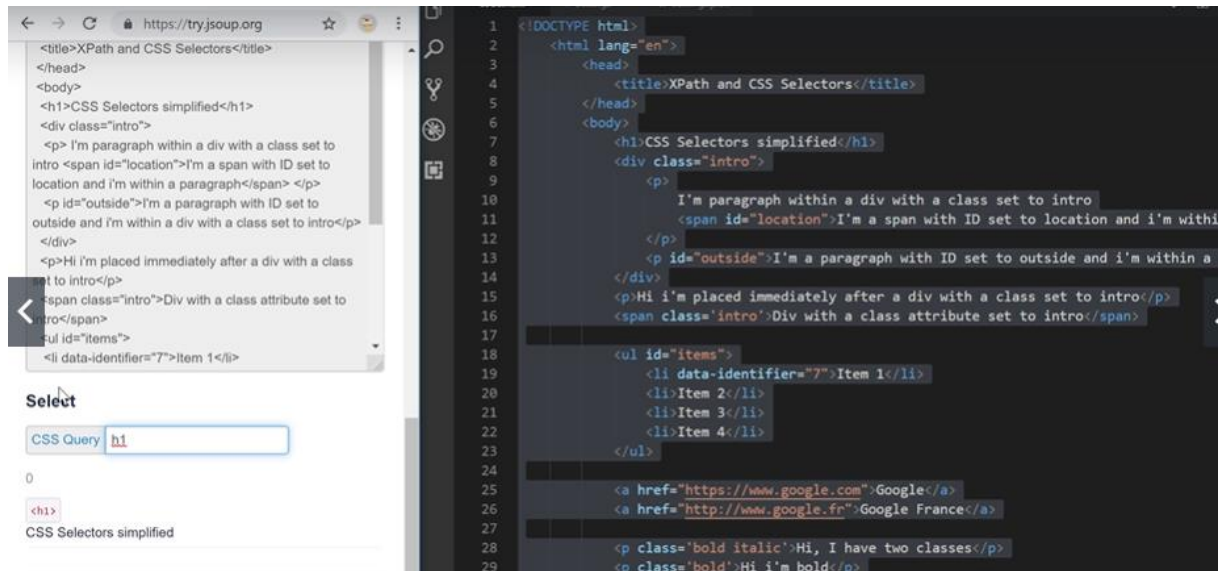
Bu kısımda örnek kodu <https://try.jsoup.org/> adresine yapıştıracağız ve kod üzerinde CSS Selectors uygulaması yapacağız yani belirli kısımları CSS Selectors ile seçmeyi öğreneceğiz.

Bu işlemi google chrome üzerinde de test edebiliyorduk, gerçek sitelerde böyle denemeler yapacağız ancak şimdilik temelleri anlamak için bu siteyi kullanalım.

Select any HTML element:

→ Just type the tag name.

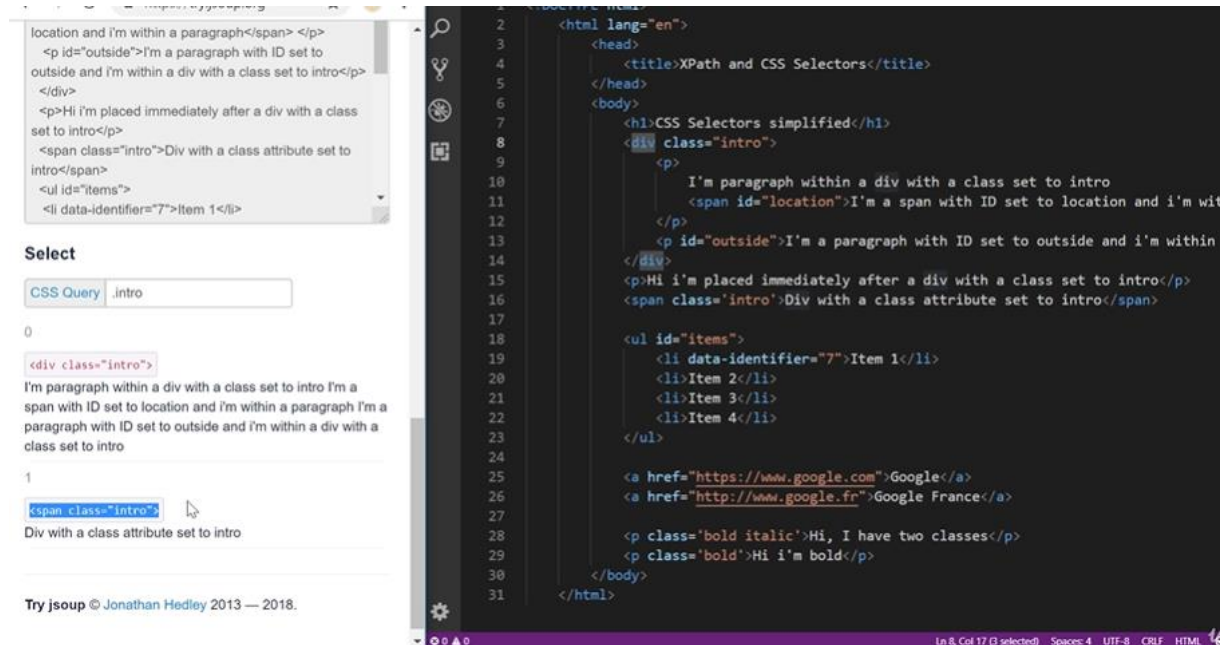
→ h1 elementini seçmek istiyorsak **h1** yazdığımızda bu eleman seçilir.



Ancak genelde daha specific selectionlar yaparız çünkü aynı tag name'i kullanan bir çok element sayfada yer alabilir.

Select elements by class name:

- `.className` ile ilgili class'a sahip olan tüm elemanları seçmiş oluruz.
- Örnek olması için "intro" class'ına sahip olan bir div ve bir span var, bunların ikisi de `.intro` komutu ile seçilebilir, anladığım kadarıyla seçilen eleman içinde olan diğer tüm yazılar da seçilmiş oluyor.



- Yani `.className` ile yapılan şey sayfada class attribute'u className olan bütün tagleri seç demek.

Select elements by ID:

- Benzer şekilde belirli bir ID attribute'una sahip olan bir elemanı seçmek istersek `#idName` ile seçim yapabiliriz.

Aynı class'a sahip elemanlar arasından, spesifik bir tagli olanı seçmek:

- Örneğin yukarıdaki `.intro` ile seçim örneğinde hem bir div hem de bir span elemanının class attribute'u intro olduğu için ikisi de seçilmişti, eğer biz sadece intro class'ına sahip olan div'leri seçmek istersek yapacağımız şey şu:

`div.intro`

Selects div elements which has class = "intro".

Aynı ID'ye sahip elemanlar arasından, spesifik bir tagli olanı seçmek:

`span#location`

Selects span elements which has ID = "location".

Birden fazla Class'a sahip olan elemanları seçme:

→ `.class1Name.class2Name`

→ Aşağıda iki tane p elemanı var ilki hem bold class'ına hem de italic class'ına sahip ikincisi sadece bold class'ına sahip, HTML elemanları için boşluk bırakarak birden fazla class tanımlayabiliyoruz.

```
<p class='bold italic'>Hi, I have two classes</p>
<p class='bold'>Hi i'm bold</p>
```

→ Burada eğer `.bold` dersem her iki p elemanını da ayrı ayrı elde ederim.

→ Sadece ilkinin elde etmek istersem `.bold.italic` diyerek iki class'ı da belirtmem gerek.

Class veya ID dışındaki attribute'lara göre eleman seçme:

→ Bir HTML elemanı class veya ID dışında başka attribute'lara da sahip olabilir, bunlardan bazıları aşağıda gösterilmiş: href veya data-identfier gibi attributelar.

```
<span class='intro'>Div with a class attribute set to intro</span>

<ul id='items'>
  <li data-identfier='7'>Item 1</li>
  <li>Item 2</li>
  <li>Item 3</li>
  <li>Item 4</li>
</ul>

<a href='https://www.google.com'>Google</a>
<a href='http://www.google.fr'>Google France</a>

<p class='bold italic'>Hi, I have two classes</p>
<p class='bold'>Hi i'm bold</p>
```

→ Eğer data-identfier attribute'u 7 olan li item'ını seçmek istersek yapmamız gereken şey:

`li[data-identfier=7]` :class attribute'u için de `[class=className]` kullanabiliriz

→ Eğer href'i `https://www.google.com` olan tüm elemanları seçmek istersek:

`[href = https://www.google.com]`

Sadece attribute'u belirli bir harf veya harf grubu ile başlayan elemanları seçme: (^)

→ Mesela href attribute'u "http" ile başlayan elemanları seçmek istedik diyelim:

`[href^='http']` veya `[href^=http]` tırnak işareti şart değil.

→ Attribute'u http ile başlayan a tagli elemanları seçmek istersek:

`a[href^='http']`

→ Aynı özelliği class veya ID attribute'ı için de kullanabileceğimizi unutma

`[class^='i']`

Sadece attribute'u belirli bir harf veya harf grubu ile sonlanan elemanları seçme: (\$)

→ Yukarıdaki örneğe çok benzer ^ işareti yerine \$ işareti kullanılır.

→ href attribute'u .fr ile biten tüm li elementlerini seçmek istersek:

`li[href$=.fr]`

Sadece attribute'u belirli bir harf veya harf grubunu içeren elemanları seçme: (*)

→ Yani diyelim ki class attribute'u içerisinde tr içeren tüm span elemanlarını seçmek istiyoruz:

`span[class*=tr]`

Belirli bir elemanın içerisindeki elemanları seçme:

→Örneğin intro class'lı div elemanının içerisinde p elemanlarını seçmek istiyorum diyelim:

```
<h1>CSS Selectors simplified</h1>
<div class="intro">
  <p>
    I'm paragraph within a div with a class set to intro
    <span id="location">I'm a span with ID set to location and i'm within
  </p>
  <p id="outside">I'm a paragraph with ID set to outside and i'm within a d
</div>
<p>Hi i'm placed immediately after a div with a class set to intro</p>
<span class="intro">Div with a class attribute set to intro</span>
```

div.intro p

→intro class'lı div elemanlarının içerisindeki tüm p elemanları seçilecek. Sadece div.intro dediğimizde bu class'ın içerisindeki tüm textler seçiliyor ancak elemanlar seçilmiyor, yani bir tag içerisinde bir başka tag varsa o tag'i ayrıca seçmek istersek bu şekilde boşlukla belirtmemiz gerek, yoksa sadece o tag içerisindeki text seçiliyor.

→Mesela bu örnekte div.intro elemanı içerisindeki p'ler ayrı ayrı seçildi ama ilk p içerisinde span seçilmedi:

Select

CSS Query div.intro p

0

<p>

I'm paragraph within a div with a class set to intro I'm a span with ID set to location and i'm within a paragraph

1

<p id="outside">

I'm a paragraph with ID set to outside and i'm within a div with a class set to intro

→ Bu span'i ayrıca seçmek için:

`div.intro p span` demeliyiz

→ Yok ayrıca seçmek istemiyoruz, div içerisindeki tüm elemanları ayrı ayrı seçmek istiyoruz, span p'nin için de olsa da onu da ayrı bir eleman olarak görmek istiyoruz diyorsan:

`div.intro p,span#location` ile ilgili span'i de specific olarak seçime dahil etmiş oluruz, virgül kullanınca iki ayrı seçimi birleştirmiş oluyoruz yani ve bağlacı !

→ Yani yukarıdaki ifade şunu diyor, bana intro class'lı div içerisindeki tüm p'leri getir, yanında bir de location ID'li span'i getir. p'nin içinde başka elemanlar varsa ben anlamam onu p'nin içeriği olarak kabul ederim.

→ Aynı seçimi şöyle de yapabiliriz:

`div.intro p,div.intro p span`

`div.intro` içerisindeki p'leri getir bir de yanında `div.intro` içerisindeki p'lerin içerisindeki spanleri getir.

Belirli bir elemanın içerisindeki elemanları seçme 2:

→ Yukarıdaki başlığa bir alternative olarak bir elemanın direct children'larını seçmek için > işaretini kullanabiliriz.

→ Örneğin intro class'ına sahip div elemanının p tagli direct children'larının hepsini seçmek istersek:

`div.intro > p` yani div intro p ile aynı şey

→ Bu p elemanlarının span tagli direct children'larını seçmek istersem de

`div.intro > p > span` derim.

→ Hem p elemanlarını hem de onun içindeki spanleri seçmek istersem:

`div.intro > p, div.intro > p > span` derim.

Bu yaklaşım yukarıdaki yaklaşıma bir alternatiften fazlası değil, aslında > işaretini sildiğin zaman birebir aynı oluyor.

Belirli bir elemanın hemen ardında konumlanan elemanı seçme:

→ Diyelim ki aşağıdaki div elemanında hemen sonra gelen p tag'li elemanı seçmek istiyorum.

```
5 <body>
6 <h1>CSS Selectors simplified</h1>
7 <div class="intro">
8   <p>
9     I'm paragraph within a div with a class set to intro
10    <span id="location">I'm a span with ID set to location and i'm within
11  </p>
12  <p id="outside">I'm a paragraph with ID set to outside and i'm within a d
13 </div>
14 <p>Hi i'm placed immediately after a div with a class set to intro</p>
15 <span class="intro">Div with a class attribute set to intro</span>
16 </body>
```

div.intro+p

div.intro elemanının hemen ardından gelen p'yi seç demek, **HEMEN ARDINDAN** gelmesi önemli!

div.intro+span dersek bir **sonuç alamayız** çünkü span div.intro'dan hemen sonra gelmiyor iki sonra geliyor.

Belirli bir elemandan sonra gelen elemanı seçme (hemen ardından şartı aranmaz):

div.intro ~ p

n. child olan tag'i seçme:

→ Aşağıki ul elemanı içerisinde 4 tane child li elemanı var, ben 1. Ve 3. Child li elemanını seçmek istersem

```
<ul id="items">
  <li data-identifier="7">Item 1</li>
  <li>Item 2</li>
  <li>Item 3</li>
  <li>Item 4</li>
</ul>
```

li:nth-child(1), li:nth-child(3)

→ Burada li'nin hangi elemanın child'ı olduğunu belirtmedik, yani bir başka elemanın child'ı olan başka li'ler olsaydı sanıyorum onları da seçecekti.

Odd veya even child'ları seçme:

li:nth-child(odd)

li:nth-child(even)

XPath Fundamentals

Bu kısımda geçen kısımdaki işlemlerin aynısını Xpath ile yapacağız. Kursta yapacağımız projelerde her zaman Xpath'i CSS selectors'a tercih edeceğiz. CSS Selectors'ı öğrendik, çünkü splash'i öğrendiğimizde CSS Selectors kullanılacak.

Geçen kısımda olduğu gibi bir kodu <https://scrapinghub.github.io/xpath-playground/> adresine yerleştireceğiz ve XPath denemelerimizi bu adreste yapacağız, gerçek projelerde ise chrome kullanacağız.

Select any HTML element:

`//elementTag`

→ Mesela h1 elemanlarını seçmek istiyorsak:

`//h1`

Select specific elements with class name:

`//div[@class='intro']` intro class'lı div elementlerini seçer.

Select child element by tag:

`//div[@class='intro']/p` intro class'lı div elementinin içerisindeki p tagli child'lar.

İki farklı class'ın child'larını OR ile seçme:

`//div[@class='intro' or @class='outro']/p` class attribute'u intro veya outro olan div elemanlarının içerisindeki p elemanlarını seçer.

CSS Selector ile XPath arasındaki bir fark:

CSS Selector ile bir eleman seçildiğinde hiyerarşik olarak onun içinde olan elemanlar görünmüyor, sadece seçilen elemanın içerisindeki text kısımları görünüyordu, XPath ile seçin yaptığımızda ise bu children tags görünür.

Eğer sadece elemanın içerisindeki text'i seçmek istersek:

Seçimin sonuna `/text()` ekleriz:

`//div[@class='intro' or @class='outro']/p/text()`

Elemanın içerisindeki bir attribute'u seçmek:

→ Mesela aşağıdaki gibi iki a elemanının da href attribute'u var diyelim ki href attribute'u olan a elemanların href attribute'larını seçmek istiyoruz.

```
<a href="https://www.google.com">Google</a>  
<a href="http://www.google.fr">Google France</a>  
</body>
```

//a/@href

→ Buna benzer şekilde class tüm HTML sayfasında class attribute'u olan tüm elemanların class attribute'larını seçmek istersek

//@class

Attribute'u bir harf veya harf grubu ile başlayan elemanları seçme:

//a[starts-with(@href,"https")] href attribute'u 'https' ile başlayan a elemanlarını seçer.

Attribute'u bir harf veya harf grubu ile biten elemanları seçme:

//a[ends-with(@href,"fr")] eski versiyonlarda bu fonksiyon sıkıntı çıkarabilir, daha sonra bundan bahsedeceğiz

Sadece attribute'u belirli bir harf veya harf grubunu içeren elemanları seçme:

//a[contains(@href,"google")]

Elemanın text içeriğine göre seçim yapmak:

//a[contains(text(),"France")] text'i içerisinde case sensitive olarak France kelimesi geçen a elemanları seçilir.

Belirli bir elemanın içerisindeki elemanları seçme:

→ Zaten yukardak bahsettik ama mesela 'items' id'li ul elemanlarının içerisindeki tüm li elemanlarını seçelim:

`//ul[@id='items']/li`



→ Görüldüğü gibi tüm li elemanları seçiliyor

Seçilen elemanlar arasından bir veya daha fazlasını seçme:

→ Mesela yukarıdaki örnekte 4 tane li elemanı seçildi, sadece ilk li elemanını seçmek istersek:

`//ul[@id='items']/li[1]`

Veya

`//ul[@id='items']/li[position()=1]`

→ Hem birinci hem de dördüncüyü seçmek istersek:

`//ul[@id='items']/li[position()=1 or position()=4]`

→ Hem birinci hem de sonuncuyu seçmek istersek:

`//ul[@id='items']/li[position()=1 or position()=last()]`

→ İkinci elemandan sonuncuya kadar seçme:

`//ul[@id='items']/li[position()>1]`

→ İkinci elemandan üçüncüye kadar seçme, sadece 2. Eleman:

`//ul[@id='items']/li[position()>1 and position()<3]`

Navigating using XPath (Going Up)

Bu ve sonraki kısımda göreceğimiz navigating örnekleri CSS Selectors'ın yapamadığı bir şey, yani XPath'in avantajlarından biri.

Önceki kısımda örneğin intro class'lı div'in child'ı olan p elementlerini nasıl seçeceğimizi gördük:

```
//div[@class='intro']/p
```

Parent Axis

Peki ya bir item'in child'ını değil de parent'ını seçmek istersek? Bunu nasıl yaparız? Mesela aşağıdaki unique ID'li p elementinin parent'ını seçmek istersek:

```
<div class="outro">
  <p id="unique">I'm in a div with a class attribute set to o
</div>
```

```
//p[@id='unique']/parent::div
```

XPath'da parent bir axis'tir. Axis HTML tree'de navigate etmek için kullanılır.

Yukarıdaki örnek için parent elementin div olduğunu biliyorduk ancak bazı örneklerde parent elementi bilmiyoruz olabiliriz, parent tag'i ne olursa olsun sadece parent'ı seçmek istiyorsak:

```
//p[@id='unique']/parent::node()
```

Ancestor Axis

Kullanımı parent'a çok benzer ancak parent ilgili elemanın bir üstünü verirken, ancestor kullanımı ile ilgili elemanın parent'ını, onun parent'ını ve varsa onun da parent'ını şeklinde tüm parentları ayrı bir eleman olarak alır.

```
//p[@id='unique']/ancestor::node()
```

Hem ilgili p elemanını hem de ancestor'ları elde etmek istersek:

```
//p[@id='unique']/ancestor-or-self::node()
```

Preceding Axis

İlgili elemandan önce gelen tüm elemanları seçer (**ancestors hariç**).

`//p[@id='unique']/preceding::node()`

Aşağıdaki örnek için ilgili elemanın ancestorları:

- `div class="outro"`
- `body`
- `html`

Bunların dışında kalan tüm elemanlar seçilir, içiçe olan elemanlar da ayrı ayrı seçilir.

- `Head, title, h1, div class="intro", p, p`

```
<!DOCTYPE html>
<html lang="en">

<head>
  <title>XPath and CSS Selectors</title>
</head>

<body>
  <h1>XPath Selectors simplified</h1>

  <div class="intro">
    <p>
      I'm paragraph within a div with a class set to intro
      <span id="location">I'm a span with ID set to location and i'm within a paragraph</span>
    </p>
    <p id="outside">I'm a paragraph with ID set to outside and i'm within a div with a class set to intro
    </p>
  </div>

  <div class="outro">
    <p id="unique">I'm in a div with a class attribute set to outro</p>
  </div>

  <p>Hi i'm placed immediately after a div</p>

  <span class="intro">Div with a class attribute set to intro</span>

  <ul id="items">
    <li data-identifier="7">Item 1</li>
    <li>Item 2</li>
    <li>Item 3</li>
    <li>Item 4</li>
  </ul>

  <a href="https://www.google.com">Google</a>
  <a href="http://www.google.fr">Google France</a>

</body>
</html>
```

Bu elemanlardan specific tag'li olanı istersek ifadenin sonuna `node()` yerine eleman tag'ini yazmamız yeterli:

`//p[@id='unique']/preceding::h1` sadece h1 precedingleri seçer.

Preceding Sibling Axis

Bu axis ile bir elemandan önce gelen kardeşlerini seçebiliriz.

Kardeş elemanlar aynı parent'a sahip olan elemanlardır, örneğin bir div'in iki p child'ı varsa ben 1.'yi seçmek için 2.den önceki sibling'i seç diyebilirim:

```
<div class="intro">
  <p>
    I'm paragraph within a div with a class set to intro
    <span id="location">I'm a span with ID set to location and i'm within a paragraph</span>
  </p>
  <p id="outside">I'm a paragraph with ID set to outside and i'm within a div with a class set to intro
</p>
</div>
```

```
//p[@id='outside']/preceding-sibling::node()
```

Navigating using XPath (Going Down)

Önceki kısımda HTML tree'de yukarı doğru hareket etmemize olanak veren 4 axis görmüştük, şimdi HTML tree'de aşağıya doğru nasıl hareket edeceğimizi göreceğiz.

Child Axis

Ki zaten daha önce yaptığımız şey de hep aşağıya doğru eleman seçmekti mesela intro class'ı bir div'in içindeki tüm p elemanlarını seçmek istersek şöyle diyorduk:

```
//div[@class='intro']/p
```

Aslında bu seçim child axis'in shortcut'ı yani yukarıdaki command şu command ile aynı:

```
//div[@class='intro']/child::p
```

Eğer spesifik bir child değil de tüm child'lara bakıyorsak:

```
//div[@class='intro']/child::node()
```

Descendant Axis

Child axis ile bir elemanın child'larını elde edebiliyorduk, descendant ile ise hem child'ları hem de torunları elde edebiliriz.

```
//div[@class='intro']/descendant::node()
```

Following Axis

Bir elemandan sonraki tüm elemanları döndürür, preceding'in tersi gibi düşün preceding'de parents hariçti burada childs hariç. Childs hariç demek element tag'i kapandıktan sonraki bütün elemanları seçmek demek.

```
//div[@class='intro']/following::node()
```

İç içe olan elemanları da yine ayrı ayrı seçiyor.

Following Sibling Axis

Bir elemandan sonra gelen tüm kardeşlerini seçmek için:

```
//div[@class='intro']/following-sibling::node()
```

3. PROJECT 1 – SPIDERS FROM A to Z

Part 1

Bu kısımda sıfırdan bir çalışan spider oluşturacağız. Farklı ülkelerin populasyonları scrape eden bir spider olacak bu. Zaten ilk kısımda kabaca bir spider oluşturmuştuk ve bir HTML sayfasından title'ı ve country links'i çekmiştik.

İlk başlıkta oluşturulan spider aşağıdaki gibiydi:

```
class CountriesSpider(scrapy.Spider):
    name = 'countries'
    allowed_domains = ['www.worldometers.info/']
    start_urls = ['https://www.worldometers.info/world-population/population-by-country/']

    def parse(self, response):
        #response üzerinden h1 etiketinin içindeki text'i çekelim:
        #burada tek selector objesi döndürülecek çünkü tek bir h1 etiketi var.
        #selector içinde xpath ve çekilen etiket datası yer alacak.
        title = response.xpath("//h1/text()").get()

        #response üzerinde td/a etiketlerinin içindeki text'i list olarak çekelim:
        #burada 233 tane selector döndürülecek.
        #çünkü 233 tane ilgili etiketten var.
        #her selector'ın datasını list olarak çekmek için getall() kullandık.
        countries = response.xpath("//td/a/text").getall()

        #Scrapy'de return edilecek datayı her zaman bir dict olarak return etmeliyiz:
        #yield'i return gibi düşün, detayın istersen bak.
        yield{
            'title':title
            'countries':countries
        }
```

Şimdi bunun üzerinde değişiklikler yapacağız, öncelikle title'ı scrape etmek istemediğimiz için bu kısmı kaldırıyoruz.

Ayrıca `//td/a/text` ile sadece td içindeki a elementlerinin içindeki text kısmı seçiliyordu biz ise bu örnekte a elementinin içindeki text'in yanında a elementinin href'ini de elde etmek istiyoruz zira bu href bizi ilgili country'nin yıllara göre populasyon dağılımını gösteren bir başka sayfaya yönlendiriyor.

Bu yüzden ilgili satırı şöyle değiştiriyoruz:

`countries = response.xpath("//td/a")` burada `getall` kullanmadık, return edilen şey 233 selector elemanı olan bir list olacaktır:

Output'u test etmek istersek terminalde bir `scrapy shell "URL"` komutu ile ilgili URL'i scrape etmek için bir shell başlatırız. Daha sonra aşağıdaki kodu yazarız ve `countries`'in içeriğine bakabiliriz.

```
n [1]: countries = response.xpath("//td/a")
n [2]: countries_
```

Sonuca bakarsak selector objelerinden oluşan bir list görürüz, bu objenin 2 attribute'u var `xpath` ve `data`, `data` kısmı doğrudan a etiketinin içeriğini tamamen içeriyor. Text deseydik sadece text kısmını alacaktık ve sonuna `getall()` koysaydık da sadece `data` kısımlarını alıp yeni bir list döndürecekti.

```
<Selector xpath="//td/a" data='<a href="/world-population/monaco-popula'>,
<Selector xpath="//td/a" data='<a href="/world-population/liechtenstein'>,
<Selector xpath="//td/a" data='<a href="/world-population/turks-and-cai'>,
<Selector xpath="//td/a" data='<a href="/world-population/gibraltar-pop'>,
<Selector xpath="//td/a" data='<a href="/world-population/san-marino-po'>,
<Selector xpath="//td/a" data='<a href="/world-population/british-virgi'>,
<Selector xpath="//td/a" data='<a href="/world-population/caribbean-net'>,
<Selector xpath="//td/a" data='<a href="/world-population/palau-populat'>,
<Selector xpath="//td/a" data='<a href="/world-population/cook-islands-'>,
<Selector xpath="//td/a" data='<a href="/world-population/anguilla-popu'>,
<Selector xpath="//td/a" data='<a href="/world-population/wallis-and-fu'>,
<Selector xpath="//td/a" data='<a href="/world-population/tuvalu-popula'>,
<Selector xpath="//td/a" data='<a href="/world-population/nauru-populat'>,
```

Sonuçta `countries = response.xpath("//td/a")` komutu ile ilgili sayfadaki a etiketlerini yukarıdaki gibi selector list içinde elde edebiliyorum.

Scrapy shell'i test için kullandık şimdi spider'a dönüp kodu gerçekleyebilirim:

Parse metodunu aşağıdaki gibi değiştirdik:

```
4
5 class CountriesSpider(scrapy.Spider):
6     name = 'countries'
7     allowed_domains = ['www.worldometers.info/']
8     start_urls = ['https://www.worldometers.info/world-population/population-by-country/']
9
10    def parse(self, response):
11        countries = response.xpath("//td/a")
12        for country in countries:
13            name = country.xpath("./text()").get()
14            link = country.xpath("./@href").get()
15
16            yield {
17                'country_name': name,
18                'country_link': link
19            }
20
```

Burada yapılan şey şu:

- Öncelikle bir selector list olarak a elemanları elde edildi.
- Daha sonra her bir selector'ı dönecek bir for loop oluşturuldu.
 - o Her bir selector objesi için tekrar xpath kullanabiliyoruz, loop'un içinde yapılan da bu ancak bu kullanımın response kullanımından küçük bir farkı var `//` yerine `./` ile başlıyoruz.
 - o Sonuçta her country içinden text kısmı yani country ismi kısmı alındı, `get()` metodunun işlevi selector list'in içindeki dataları alıp string olarak döndürmek tek bir selector objeden oluşan list söz konusuysa tek bir string output verecektir. Get demeseydik yine data kısmı text olan bir selector objesi elde edecektik.
 - o Link satırında yapılan şey ise selector objesinin datasında tutulan a elementinin text'i yerine href attribute'unu string olarak almak.

Aşağıda `text()`'in ve `.get()`'in işlevini anlayabiliriz:

```
countries=response.xpath("//td/a")
countries[0]
<Selector xpath='//td/a' data='<a href="/world-population/china-popu...'>
countries[0].xpath("./text()")
[<Selector xpath='./text()' data='China'>]
countries[0].xpath("./text()").get()
'China'
```

Sonuçta yukarıdaki countries spider'ını VS Code üzerinden terminalden `scrapy crawl countries` diyerek çalıştırabiliriz ve sayfadaki her country name'ini ve linkini elde edebiliriz.

yield'in yaptığı şey sanıyorum parse fonksiyonunu bir generator'a dönüştürmek daha sonra parse generator'ı içinde iterate edilip her iteration'da bir dictionary elde edilebilir her dictionary country name ve link içerecek. Bunun detayına kursta girmedim şimdilik böyle kabul et gerekirse tekrar bakarsın.

Part 2

Şimdi önceki kısımda öyle ya da böyle ana URL'den ülke isimlerini ve karşılık gelen link'i elde ettik, ancak bizim amacımız bu linklerin içine girip o ülke sayfasından ülkeyle ilgili daha fazla bilgi çekmek olabilir.

Anladığım kadarıyla bu spider'ın start_urls'inde belirtilen adrese zaten bir request gönderildi ve parse içinde bu request'in response'u handle ediliyordu ancak şimdi, bu response'dan elde edilen linklere yeni request'ler göndermemiz gerekiyor.

Yani her ülke için o ülkenin linki/URL'sine bir request göndereceğiz, shell içinde bu request gönderme işlemini `fetch(URL)` komutu ile veya önce bir request object oluşturduktan sonra fetch'e besleyerek yani `fetch(scrapy.request(URL))` komutu ile yapabiliyorduk.

Ancak bu fetch komutu shell için tanımlı, spider içinde bunu kullanamıyoruz.

Özetle amacımız her country linkine bir request göndermek:

Bunun için önceki parse metodundaki yield kısmını değiştireceğiz, metodun önceki hali aşağıdaki gibi:

```
4
5 class CountriesSpider(scrapy.Spider):
6     name = 'countries'
7     allowed_domains = ['www.worldometers.info/']
8     start_urls = ['https://www.worldometers.info/world-population/population-by-country/']
9
10    def parse(self, response):
11        countries = response.xpath("//td/a")
12        for country in countries:
13            name = country.xpath("./text()").get()
14            link = country.xpath("./@href").get()
15
16            yield {
17                'country_name': name,
18                'country_link': link
19            }
20
```

Spider içinde yield kısmını silip aşağıdaki gibi değiştirdiğimizde her country url için bir request göndermeyi umud ediyoruz:

yield scrapy.Request(url=link)

```
def parse(self, response):
    countries = response.xpath("//td/
    for country in countries:
        name = country.xpath("//text
        link = country.xpath("//@hre
        yield scrapy.Request(url=link)
```

Ancak bu yöntemle **missing scheme in request** hatası alıyoruz bunun sebebi şu, a elementleri içinden elde edilen linkler absolute linkler değil **relative linkler** yani mesela China'ya karşılık gelen link: /world-population/china-population/ şeklinde e biz de doğrudan bu adrese request göndermeye çalıştığımız için hata alıyoruz. Bu hata ilgili linkin http veya https protokollerinden hangisine dahil olduğu bilinmediği zaman alınır.

```
align:left">
<a href="/world-population/china-population/"
China/as
</td>
```

Bu problemi çözmek için yapacağımız şey bu relative linkleri domain address ile concatenate edip sonuçta request gönderebileceğimiz absolute linkler elde etmek.

Bunu yapmak için aşağıdaki gibi f-string kullanıyorum yani yapılan şey basitçe worldometers.info string'i ile link stringini concat etmek ve sonuçta request bu adrese gönderiliyor.

```
def parse(self, response):
    countries = response.xpath("//td/a")
    for country in countries:
        name = country.xpath("//text()").get()
        link = country.xpath("//@href").get()

        absolute_url = f"https://www.worldometers.info{link}"

        yield scrapy.Request(url=absolute_url)
```

Bu sefer de **filtered offsite request hatası** aldık bu hatayı düzeltmek için basitçe spider'in allowed_domains attribute'u içindeki URL'in sonundaki /'i sildik. Bu hata extra slash'lerden dolayı olur diyor.

Bu düzeltmeden sonra gördük ki scrapy tek tek linklere request gönderiyor, bunu durdurmak için **CTRL+C**'ye iki kez basıyoruz.

Alternative

Yukarıdaki yaklaşım, country linklerine request göndermek için gayet iyi çalışıyor ancak best practice değil. Aynı işlemi yapmak için bir başka yöntem urljoin function'ını kullanmak:

```
def parse(self, response):
    countries = response.xpath("//td/a")
    for country in countries:
        name = country.xpath("./text()").get()
        link = country.xpath("./@href").get()

        # absolute_url = f"https://www.worldometers.info{link}"
        absolute_url = response.urljoin(link)

        yield scrapy.Request(url=absolute_url)
```

Burada urljoin fonksiyonu spider'ın request ettiği sayfa linki ile elde edilen relative linki concat ediyor, böylece biz domain'i tekrar yazmamış oluyoruz, daha temiz bir görüntü.

Another Alternative

Eğer ki relative link'leri absolute link'e çevirmek için bir şey yapmak istemiyorsak aşağıdaki versiyonu kullanabiliriz. Anladığım kadarıyla bunun da yaptığı ilgili response'un request URL'sini alıp relative linkle concat etmek ve sonra bu linke bir request göndermek. Bu iki işlem tek fonksiyonda yapılmış oldu.

```
def parse(self, response):
    countries = response.xpath("//td/a")
    for country in countries:
        name = country.xpath("./text()").get()
        link = country.xpath("./@href").get()

        # absolute_url = f"https://www.worldometers.info{link}"
        # absolute_url = response.urljoin(link)

        yield response.follow(url=link)
```

Sonuçta final spiderımız şu şekilde oldu:

```
class CountriesSpider(scrapy.Spider):
    name = 'countries'
    allowed_domains = ['www.worldometers.info']
    start_urls = ['https://www.worldometers.info/world-population/population-by-country/']

    def parse(self, response):
        countries = response.xpath("//td/a")
        for country in countries:
            name = country.xpath("./text()").get()
            link = country.xpath("./@href").get()

            # absolute_url = f"https://www.worldometers.info{link}"
            # absolute_url = response.urljoin(link)

            yield response.follow(url=link)
```

Bu şekilde artık her country linkine bir request gönderebiliyoruz. Sonraki kısımda ise bu request'lerin response'larını nasıl alacağımızı göreceğiz.

Part 3

Önceki kısımda countries spider' ile worldometers.info url'sine gönderilen request sonucu alınan response'da country name'leri ve karşılık gelen linkleri elde etmenin yanında her bir country linkine bir request yollamıştık.

Şimdi ise bu yollanan request'lerin response'larını nasıl yakalayacağımızı göreceğiz.

Bu amaç için yapmamız gereken şey response.follow veya scrapy.Request'e bu gönderilen request'lerin response'larını nereye göndereceğini söylemek eğer bunu söylersek, response'u nereden yakalayacağımızı da bilmiş oluruz.

Bu amaçla spider içerisinde yeni bir method yaratacağız, ve response.follow ya da eğer kullandıysak scrapy.Request methodu içerisine bir callback tanımlayacağız:

```
def parse(self, response):
    countries = response.xpath("//td/a")
    for country in countries:
        name = country.xpath("./text()").get()
        link = country.xpath("./@href").get()

        # absolute_url = f"https://www.worldometers.info/{link}"
        # absolute_url = response.urljoin(link)

        yield response.follow(url=link, callback=self.parse_country)

def parse_country(self, response):
```

Böylece yapılan işlem şu oldu, response.follow ile bir linke request gönderildiğinde bu request'in response'u parse_country methoduna gönderilsin dendi. Artık her request'in response'una ne yapılacağını parse_country method içerisinde karar verebiliriz.

Peki callback'in olayı ne? Callback'in olayı şu, önce request yapılacak, request tamamlandıktan emin olunduktan sonra response callback function'a geçirilecek, callback kullanmak yerine iki fonksiyonu ardarda yazarsak sanırım, request sonuçlanmadan, response alınmaya çalışılabilir bu da problem demek. Çünkü request'in tamamlanması fonksiyonun execute edilmesinden bağımsız bir olay eğer sitede bir sorun varsa request fonksiyonu execute edilse de response alınamayacak, callback sayesinde response alındığından emin olunduktan sonra callback fonksiyonu çağırılıyor.

callback is a function that is to be executed after another function has finished executing — hence the name 'call back'. Callbacks are a way to make sure certain code doesn't execute until other code has already finished execution.

Daha sonra `parse_country` methodunun içine `logging.info(response.url)` böylece her request gönderildikten sonra alınan response URL ile ilgili bir info log edilecek.

Spider bu haldeyken **scrapy crawl countries** ile spider'ı çalıştırınca görüyoruz ki ekran sürekli akıyor ve her gönderilen requestten sonra bir info ekrana yazdırılıyor. Zaten bunu istemiştik.

Year	Population (millions)	Yearly % Change	Yearly Change	Migrants (net)	Median Age	Fertility Rate	Density (P/Km ²)
2019	1,420,062,022	0.35 %	5,016,094	-324,919	37.3	1.61	150
2018	1,415,045,928	0.39 %	5,528,531	-324,919	37.3	1.61	150
2017	1,409,517,397	0.43 %	6,017,032	-324,919	37.3	1.61	150
2016	1,403,500,365	0.46 %	6,471,812	-324,919	37.3	1.61	150
2015	1,397,028,553	0.54 %	7,454,690	-339,690	37.0	1.60	150
2010	1,359,755,102	0.57 %	7,626,322	-478,179	35.2	1.58	140
2005	1,321,623,490	0.59 %	7,684,904	-406,017	32.7	1.55	130
2000	1,283,198,970	0.69 %	8,651,793	-76,678	30.1	1.51	120
1995	1,239,940,004	1.13 %	13,498,961	-152,825	27.3	1.90	110
1990	1,172,445,200	1.83 %	20,316,362	-84,009	24.9	2.73	100
1985	1,070,863,389	1.50 %	15,397,216	-40,168	23.6	2.55	90
1980	993,877,310	1.54 %	14,586,445	-9,438	21.9	3.00	80

Gördüğümüz gibi 2019 yılı bir td elemanının içinde text olarak duruyor, bu td elemanı da bir tr elemanının içinde, tr elemanının açılımı table row elemanı yani yukarıdaki table'ın tek bir row'u bu markup içinde yer alıyor.

Bu noktada yapmak istediğimiz şey şu XPath kullanarak tüm tr'leri seçmek daha sonra loop ile bunları dönmek ve her yıl için year ve population bilgisini extract etmek.

Bunu yapmak için öncelikle XPath ifadelerini deneyerek istediğim elemanları seçip seveceğime bakıyorum `//tr` ifadesini kullanırsak yukarıdaki başka bir grafiği seçiyor, bizim bu tabloyu seçmek için daha spesifik bir XPath'e ihtiyacımız var.

Year	Population	Yearly % Change	Yearly Change	Migrants (net)	Median Age	Fertility Rate	Density (P/Km ²)
2019	1,420,062,022	0.35 %	5,016,094	-324,919	37.3	1.61	151
2018	1,415,045,928	0.39 %	5,528,531	-324,919	37.3	1.61	151
2017	1,409,517,397	0.43 %	6,017,032	-324,919	37.3	1.61	151
2016	1,403,500,365	0.46 %	6,471,812	-324,919	37.3	1.61	151
2015	1,397,028,553	0.54 %	7,454,690	-339,690	37.0	1.60	151
2010	1,359,755,102	0.57 %	7,626,322	-478,179	35.2	1.58	151
2005	1,321,623,490	0.59 %	7,684,904	-406,017	32.7	1.55	151
2000	1,283,198,970	0.69 %	8,651,793	-76,678	30.1	1.51	151

Bunun için ilgili table'ın class'ını yukarıda görüldüğü gibi kopyalıyoruz, XPath'e bunu dahil edeceğiz:

`//table[@class='copiedClassName']` bunu yaptığımızda da iki tane aynı class'tan table seçildiğini gördük biz birincisini istediğimiz için şöyle diyoruz:

`(//table[@class='copiedClassName'])[1]`

Böylece bu table'ı seçebiliyoruz:

Year	Population	Yearly % Change	Yearly Change	Migrants (net)	Median Age	Fertility Rate	Density (P/Km ²)
2019	1,420,062,022	0.35 %	5,016,094	-324,919	37.3	1.61	151
2018	1,415,045,928	0.39 %	5,528,531	-324,919	37.3	1.61	151
2017	1,409,517,397	0.43 %	6,017,032	-324,919	37.3	1.61	151
2016	1,403,500,365	0.46 %	6,471,812	-324,919	37.3	1.61	151
2015	1,397,028,553	0.54 %	7,454,690	-339,690	37.0	1.60	151
2010	1,359,755,102	0.57 %	7,626,322	-478,179	35.2	1.58	151
2005	1,321,623,490	0.59 %	7,684,904	-406,017	32.7	1.55	151
2000	1,283,198,970	0.69 %	8,651,793	-76,678	30.1	1.51	151

Yapmak istediğimiz şey bu table'ın içindeki tüm tr'leri elde etmek, daha sonra bu tr'leri loop'la dönüp her tr'den yılı ve popülasyonu gösteren td'yi çekmek.

İlgili tablodan tr'leri seçmek için önce table'ın child'ı tbody'i oradan da tbody'nin children'ı olan tr'leri seçeceğim:

```
(//table[@class='copiedClassName'])[1]/tbody/tr
```

Bu sayede ilgili tablonun her bir satır elemanı olan tr'yi seçmiş olduk yani burada China için 17 satır var, daha sonra her satırdan istediğimiz td'yi çekeceğiz.

Population of China (2019 and historical)

Year	Population	Yearly % Change	Yearly Change	Migrants (net)	Median Age	Fertility Rate	Density (P/K
2019	1,420,062,022	0.35 %	5,016,094	-324,919	37.3	1.61	
2018	1,415,045,928	0.39 %	5,528,531	-324,919	37.3	1.61	
2017	1,409,517,397	0.43 %	6,017,032	-324,919	37.3	1.61	
2016	1,403,500,365	0.46 %	6,471,812	-324,919	37.3	1.61	
2015	1,397,028,553	0.54 %	7,454,690	-339,690	37.0	1.60	
2010	1,359,755,102	0.57 %	7,626,322	-478,179	35.2	1.58	
2005	1,321,623,490	0.59 %	7,684,904	-406,017	32.7	1.55	
2000	1,283,198,970	0.69 %	8,651,793	-76,678	30.1	1.51	
1995	1,239,940,004	1.13 %	13,498,961	-152,825	27.3	1.90	
1990	1,172,445,200	1.83 %	20,316,362	-84,009	24.9	2.73	
1985	1,070,863,389	1.50 %	15,397,216	-40,168	23.6	2.55	
1980	993,877,310	1.54 %	14,586,445	-9,438	21.9	3.00	
1975	920,945,083	2.23 %	19,231,325	-216,297	20.4	4.77	
1970	824,788,457	2.68 %	20,445,255	-32,058	19.3	6.25	
1965	722,562,183	1.90 %	12,975,208	-44,170	22.3	6.03	
1960	657,686,143	1.49 %	9,370,349	-11,907	21.4	5.40	
1955	610,834,396	1.96 %	11,283,024	-44,170	22.3	6.03	

Şimdilik buraya kadarki kısmı spider üzerinde gerçekleyelim:

```
def parse(self, response):
    countries = response.xpath("//td/a")
    for country in countries:
        name = country.xpath("./text()").get()
        link = country.xpath("./@href").get()

        # absolute_url = f"https://www.worldometers.info/{link}"
        # absolute_url = response.urljoin(link)

        yield response.follow(url=link, callback=self.parse_country)

def parse_country(self, response):
    rows = response.xpath("(//table[@class='table table-striped table-bordered table-hover table-condensed table-11])")
    for row in rows:
        year = row.xpath("./td[1]/text()").get()
        population = row.xpath("./td[2]/strong/text()").get()
        yield {
            'year': year,
            'population': population
        }
```

- Her country link response için denediğimiz xpath expression'ı kullanarak 17 selector elemanı olan bir rows listesi yaratılıyor.
- Daha sonra bu rows listesi içinde gezilerek her bir selector elemanı row olarak seçiliyor.
- Bu row elemanı bir selector olduğu için bunun üzerinde tekrar xpath ile select işlemi yapabildiğimizi biliyoruz.
- Her row'dan yukarıdaki xpath ile year ve population bilgisi çekiliyor, burada population text'i td içindeki bir strong elemanının içinde olduğu için xpath'e strong eklendi.
- Get methodu kullanılmadığı sürece elde edilen sonuç bir selector object oluyor get method selectorun datasını string olarak döndürüyor.
- Sonuçta her requestten sonra ülke sayfasının response'u alınıyor sayfadaki tablodan tüm satırlar dönülüyor ve her satır için year ve population bilgisi bir dictionary olarak yield ediliyor. Her country URL reponse için 17 yıl olduğu için 17 tane dictionary yield ediliyor.

```
on/caribbean-netherlands-population/>
{'year': '2010', 'population': '20,940'}
2019-08-22 21:42:03 [scrapy.core.scrapers] DEBUG: Scraped from <200 https://www.worldometers.info/world-population/caribbean-netherlands-population/>
{'year': '2005', 'population': '14,403'}
2019-08-22 21:42:03 [scrapy.core.scrapers] DEBUG: Scraped from <200 https://www.worldometers.info/world-population/caribbean-netherlands-population/>
{'year': '2000', 'population': '14,393'}
2019-08-22 21:42:03 [scrapy.core.scrapers] DEBUG: Scraped from <200 https://www.worldometers.info/world-population/caribbean-netherlands-population/>
{'year': '1995', 'population': '15,107'}
2019-08-22 21:42:03 [scrapy.core.scrapers] DEBUG: Scraped from <200 https://www.worldometers.info/world-population/caribbean-netherlands-population/>
{'year': '1990', 'population': '13,019'}
2019-08-22 21:42:03 [scrapy.core.scrapers] DEBUG: Scraped from <200 https://www.worldometers.info/world-population/caribbean-netherlands-population/>
{'year': '1985', 'population': '12,025'}
2019-08-22 21:42:03 [scrapy.core.scrapers] DEBUG: Scraped from <200 https://www.worldometers.info/world-population/caribbean-netherlands-population/>
{'year': '1980', 'population': '11,210'}
```

Gönderilen her request için yıl ve populasyon bilgisini elde etmeyi başardık, burada yield konusu hala biraz muğlak ancak zamanla oturacağını düşünüyorum.

Sıradaki kısımda country name'i de işin içine dahil edeceğiz.

Part 4

Son spider kodumuz aşağıdaki gibiydi, sonuçta her country URL request'in response'u parse_country içinde yakalanıyor ve response üzerinden xpath ile tablo'ya ulaşıyor bu tablodan da ilgili country'nin yıllara göre popülasyonuna ulaşıyor ve her country için 17 farklı year-population dictionary'si yield ediliyor.

```
name = 'countries'
allowed_domains = ['www.worldometers.info']
start_urls = ['https://www.worldometers.info/world-population/population-by-country/']

def parse(self, response):
    countries = response.xpath("//td/a")
    for country in countries:
        name = country.xpath("./text()").get()
        link = country.xpath("./@href").get()

        # absolute_url = f"https://www.worldometers.info{link}"
        # absolute_url = response.urljoin(link)

        yield response.follow(url=link, callback=self.parse_country)

def parse_country(self, response):
    rows = response.xpath("//table[@class='table table-striped table-bordered table-hover table-c")
    for row in rows:
        year = row.xpath("./td[1]/text()").get()
        population = row.xpath("./td[2]/strong/text()").get()
        yield {
            'year': year,
            'population': population
        }
```

Şimdi bu yielding işlemine year ve population'ın yanında bir de country name'i de eklemek istiyorum böylece sonuçta country-year-population bilgilerini veren dictionary'ler elde edeceğim ve bu dictionary'leri alınca aslında elimde ülkelerin yıllara göre nüfus dağılımını veren bir dataset olacak.

Global Variable Kullanmak İşe Yaramaz:

Peki country name'ı bu yield içine nasıl dahil edeceğim? İlk akla gelen fikir country_name şeklinde bir global variable tanımlamak ve parse içinde bu değişkene ismi atayarak daha sonra parse_country içinde de bu değişkeni doğrudan kullanmak ancak bu yöntem çalışmaz. Sadece tek bir country name kullanılır.

```
name = ''
allowed_domains = ['www.worldometers.info']
start_urls = ['https://www.worldometers.info/world-population/population-by-country/']

country_name = ''

def parse(self, response):
    countries = response.xpath("//td/a")
    for country in countries:
        name = country.xpath("../text()").get()
        self.country_name = name
        link = country.xpath("../@href").get()

        # absolute_url = f"https://www.worldometers.info/{link}"
        # absolute_url = response.urljoin(link)

        yield response.follow(url=link, callback=self.parse_country)

def parse_country(self, response):
    rows = response.xpath("//table[@class='table table-striped table-bordered table-hover table-condensed tal")
    for row in rows:
        year = row.xpath("../td[1]/text()").get()
        population = row.xpath("../td[2]/strong/text()").get()
        yield {
            'name': self.country_name,
            'year': year,
            'population': population
        }
```

Send data or Sync data between two parse methods:

Bu işlem için **request meta** kullanılacak, request meta: request ile birlikte bir methoddan diğerine veri geçirmeyi sağlayan bir dictionary!

scrapy.Request() veya **response.follow()** methodlarının ikisi de **meta** argumentine sahiptir! Bu argumentin içine bir dictionary verirsek bu dictionary'e **catch** edilen **response** üzerinden ulaşabiliriz, yani bu dictionary request ile birlikte gönderiliyor ve response üzerinden ulaşılabilir.

Yani hiç global variable kullanmadan, spider'ın orjinal hali aşağıdaki gibiydi:

```
name = 'countries'
allowed_domains = ['www.worldometers.info']
start_urls = ['https://www.worldometers.info/world-population/population-by-country/']

def parse(self, response):
    countries = response.xpath("//td/a")
    for country in countries:
        name = country.xpath("./text()").get()
        link = country.xpath("./@href").get()

        # absolute_url = f"https://www.worldometers.info/{link}"
        # absolute_url = response.urljoin(link)

        yield response.follow(url=link, callback=self.parse_country)

def parse_country(self, response):
    rows = response.xpath("//table[@class='table table-striped table-bordered table-hover table-c")
    for row in rows:
        year = row.xpath("./td[1]/text()").get()
        population = row.xpath("./td[2]/strong/text()").get()
        yield {
            'year': year,
            'population': population
        }
```

Şimdi bunun üzerinden parse'ın response.follow'ı içerisinde ve parse country içerisinde aşağıdaki değişiklikleri yaptık:

```
yield response.follow(url=link, callback=self.parse_country, meta={'country_name': name})

def parse_country(self, response):
    name = response.request.meta['country_nale']
    rows = response.xpath("//table[@class='table table-striped table-bordered table-hover table-condensed t")
    for row in rows:
        year = row.xpath("./td[1]/text()").get()
        population = row.xpath("./td[2]/strong/text()").get()
        yield {
            'country_name': name,
            'year': year,
            'population': population
        }
```

country_nale değil country_name olacak!

scrapy crawl countries ile countries spider'ını çalıştırdığımızda aşağıdaki gibi bir sonuç görüyoruz her bir ülke için 17 tane dictionary output edilmiş her country name, year ve population bilgisini içeriyor.

```
{'country_name': 'United States', 'year': '1970', 'population': '209,588,150'}
2019-08-22 21:55:54 [scrapy.core.scrapers] DEBUG: Scraped from <200 https://www.worldometers.info/world-population/us-population/>
{'country_name': 'United States', 'year': '1965', 'population': '199,815,540'}
2019-08-22 21:55:54 [scrapy.core.scrapers] DEBUG: Scraped from <200 https://www.worldometers.info/world-population/us-population/>
{'country_name': 'United States', 'year': '1960', 'population': '186,808,228'}
2019-08-22 21:55:54 [scrapy.core.scrapers] DEBUG: Scraped from <200 https://www.worldometers.info/world-population/us-population/>
{'country_name': 'United States', 'year': '1955', 'population': '171,783,842'}
2019-08-22 21:55:54 [scrapy.core.scrapers] DEBUG: Scraped from <200 https://www.worldometers.info/world-population/india-population/>
{'country_name': 'India', 'year': '1990', 'population': '870,133,480'}
2019-08-22 21:55:54 [scrapy.core.scrapers] DEBUG: Scraped from <200 https://www.worldometers.info/world-population/india-population/>
{'country_name': 'India', 'year': '1985', 'population': '781,666,671'}
2019-08-22 21:55:54 [scrapy.core.scrapers] DEBUG: Scraped from <200 https://www.worldometers.info/world-population/india-population/>
{'country_name': 'India', 'year': '1980', 'population': '696,783,517'}
2019-08-22 21:55:54 [scrapy.core.scrapers] DEBUG: Scraped from <200 https://www.worldometers.info/world-population/india-population/>
{'country_name': 'India', 'year': '1975', 'population': '621,301,720'}
2019-08-22 21:55:54 [scrapy.core.scrapers] DEBUG: Scraped from <200 https://www.worldometers.info/world-population/india-population/>
{'country_name': 'India', 'year': '1970', 'population': '553,578,513'}
```

4. Building Datasets

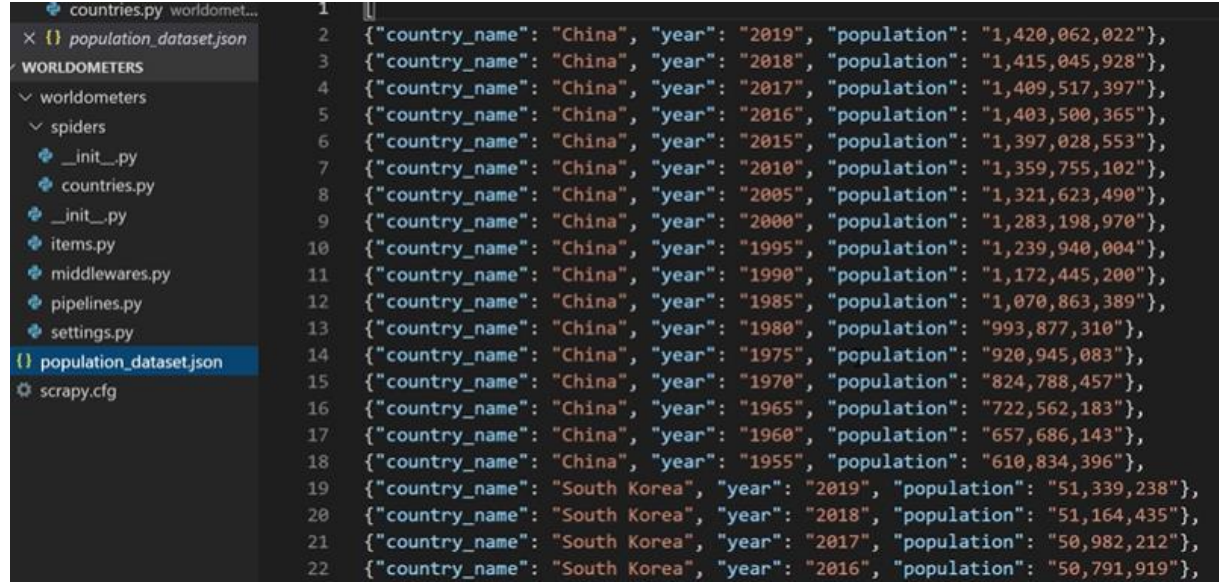
Bu kısım diğerlerine göre çok daha kısa olacak, geçtiğimiz kısmın sonunda her bir ülke için 17 tane country-year-population dictionary'si output elde etmiştik, fakat bu dictionary'i command window'da görüntülemiştik, bu output edilen datayı nasıl bir dataset olarak extract ederiz burada bunu göreceğiz.

Yani scrape edilen data'yı nasıl Json, XML veya CSV olarak export edebileceğimizi göreceğiz:

Yapacağımız şey, vs code'da terminali açmak veya ilgili environment'dan terminale ulaşip cd'yi config file'ın bulunduğu file olarak ayarlamak yani aynı spider'ı çalıştırmak ister gibi terminale ulaşmak daha sonra, spider'ı çalıştırmamanın yanına aşağıdaki gibi -o filename.extension eklemek:

```
scrapy crawl countries -o population_dataset.json
```

Böylece yield edilen dictionaries bir json file'ı içine kaydedilecek. Bu file otomatik olarak ilgili klasörde oluşturulmuş olacak, sonuçta bu file'ın içinde bir liste olarak tüm output edilen dictionary'ler yer alıyor:



```
1  {}
2  {"country_name": "China", "year": "2019", "population": "1,420,062,022"},
3  {"country_name": "China", "year": "2018", "population": "1,415,045,928"},
4  {"country_name": "China", "year": "2017", "population": "1,409,517,397"},
5  {"country_name": "China", "year": "2016", "population": "1,403,500,365"},
6  {"country_name": "China", "year": "2015", "population": "1,397,028,553"},
7  {"country_name": "China", "year": "2010", "population": "1,359,755,102"},
8  {"country_name": "China", "year": "2005", "population": "1,321,623,490"},
9  {"country_name": "China", "year": "2000", "population": "1,283,198,970"},
10 {"country_name": "China", "year": "1995", "population": "1,239,940,004"},
11 {"country_name": "China", "year": "1990", "population": "1,172,445,200"},
12 {"country_name": "China", "year": "1985", "population": "1,070,863,389"},
13 {"country_name": "China", "year": "1980", "population": "993,877,310"},
14 {"country_name": "China", "year": "1975", "population": "920,945,083"},
15 {"country_name": "China", "year": "1970", "population": "824,788,457"},
16 {"country_name": "China", "year": "1965", "population": "722,562,183"},
17 {"country_name": "China", "year": "1960", "population": "657,686,143"},
18 {"country_name": "China", "year": "1955", "population": "610,834,396"},
19 {"country_name": "South Korea", "year": "2019", "population": "51,339,238"},
20 {"country_name": "South Korea", "year": "2018", "population": "51,164,435"},
21 {"country_name": "South Korea", "year": "2017", "population": "50,982,212"},
22 {"country_name": "South Korea", "year": "2016", "population": "50,791,919"},
```

Bu dosyayı daha readable bir formata dönüştürmek için ALT+SHIFT+F i kullanabiliriz.

Json yerine CSV veya XML olarak dosya kaydetmek için sadece extension'ı değiştireceğiz, yani csv dosyası elde etmek için:

```
scrapy crawl countries -o population_dataset.csv
```

```
1 |country_name,year,population
2 |China,2019,"1,420,062,022"
3 |China,2018,"1,415,045,928"
4 |China,2017,"1,409,517,397"
5 |China,2016,"1,403,500,365"
6 |China,2015,"1,397,028,553"
7 |China,2010,"1,359,755,102"
8 |China,2005,"1,321,623,490"
9 |China,2000,"1,283,198,970"
10 |China,1995,"1,239,940,004"
11 |China,1990,"1,172,445,200"
12 |China,1985,"1,070,863,389"
13 |China,1980,"993,877,310"
14 |China,1975,"920,945,083"
15 |China,1970,"824,788,457"
16 |China,1965,"722,562,183"
```

Benzer şekilde XML için de

```
scrapy crawl countries -o population_dataset.xml
```

 diyoruz.