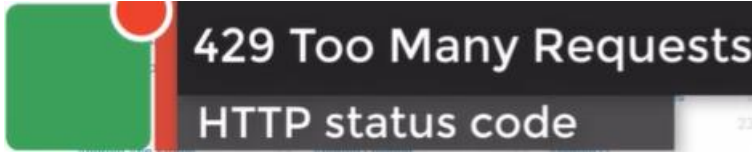


## Avoid Getting Banned Part 1 – Reasons for Getting Banned:

Spider'ımız çalışırken geçici veya kalıcı olarak ban yiyebilir.

Örneğin, serverin 429 HTTP Status kodunu geri göndermesi, too many requests anlamına geliyor.

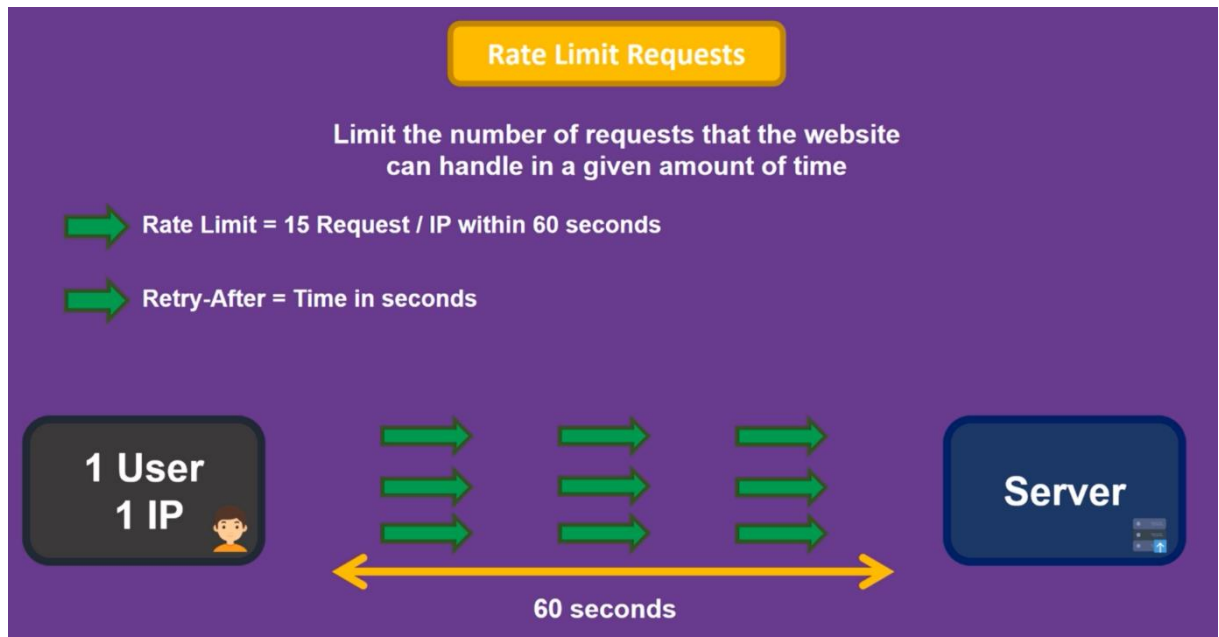


Bu kısımda öncelikle websitelerinin içeriklerini korumak için kullandığı yöntemleri anlayalım, daha sonra ileriki başlıklarda bu savunma yöntemlerini nasıl atlatabileceğimizi anlayacağız.

### Rate Limit Requests

Siteler bu yöntemi kullanarak botları engellemeyi amaçlar. Basitçe websitesinin belirli bir sürede aynı IP adresinden gelen handle edebileceği request sayısı sınırlıdır.

Örneğin: 15 Requests/IP withing 60 seconds limiti olan bir websitesine aynı IP adresinden dakikada 15'den fazla request gelirse, websitesi ilgili IP adresine temporary ban atar.



Peki bu banı nasıl atar?

- By injecting a new value to the response headers called "retry after".
- Ayrıca http STATUS CODE 429 döndürülür.

Ban yedikten sonra gönderilen request'in içindeki response headers'a bakıyoruz sonuçta aşağıdaki gibi bir retry-after:300 ibaresi görüyoruz. Yani 300 saniye ban yedin demek oluyor.

Name	× Headers Preview Response Cookies >>
categories/	<code>date: Thu, 27 Sep 2018 21:54:20 GMT</code> <code>expect-ct: max-age=604800, report-uri="https://report-uri.cloudflare.com/cdn-cgi/beacon/expect-ct"</code> <code>retry-after: 300</code> <code>server: cloudflare</code>

### Ban based on User-Agent

Websitelerinin botları engellemek için kullandığı bir diğer yöntem ise User-Agent request header'ını kontrol etmek, websiteleri bilinmeyen User-Agent header'ları ile gelen request'leri deny edebilirler.

Bu konudan zaten daha önce bahsedildi ve User-Agent header'ı nasıl değiştireceğimizi biliyoruz. İlerleyen başlıklarda da her request için random olarak farklı farklı User-Agent header'lar göndermeyi göreceğiz.

## Detection through Honeypots

Son olarak, websitelerinin botları engellemek için kullandığı bir diğer teknik olan “Detection through Honeypots” tekniğinden bahsedilecek.



Anladığım kadarıyla siteler yukarıdaki gibi linkler hidden linkler kullanırlar, bunlar bir nevi traptır.

Bu linkleri normal kullanıcılar göremez. Eğer bu bir ip’den bu invisible linke request gelirse. Websitesi bu IP’yi banlar.

---

Sonuçta botlardan korunmak amacıyla websiteleri yukarıdaki gibi yöntemleri kullanıyor olabilirler, ayrıca yine daha önce bahsedildiği gibi websitesinin robots.txt file’ına bakarak, sitenin hangi sayfaların scrape edilmesini istemediğini öğrenebiliriz.

## Avoid Getting Banned Part 2

Websiteslerinin botları engellemek amacıyla nasıl yöntemler uyguladığını önceki bölümde gördük, peki:

How can we prevent getting blacklisted by websites while scraping them ?

What are the best practices of web crawling?

### RULE 1

#RULE 1

#### DO NOT HIT WEBSITES TOO HARD

Websitesine belirli bir sürede çok fazla request göndermekten kaçınmalıyız. Çünkü websitesi bunu şüpheli bir davranış olarak kabul edip IP'mize ban atabilir.

Veya kısa sürede çok fazla request gönderilmesi yani high bandwidth sebebiyle web service unresponsive bir hale geçebilir. (DDOS Attack)

Peki spider'ımızın bu rule'a obey etmesini nasıl sağlayabiliriz?

#SOLUTION 1

- ➡ CONCURRENT\_REQUESTS = 16
- ➡ Depends on your host machine performance
- ➡ No predefined value
- ➡ For optimum performance :  $80\% \leq \text{CPU USAGE} \leq 90\%$

Birinci yol, settings.py içerisindeki `CONCURRENT_REQUESTS` variable'ının default değeri olan 16'yı değiştirerek scrapy'nin davranışını etkileyebiliriz.

Bu değer neyi temsil eder → The maximum number of concurrent (i.e. simultaneous) requests that will be performed by the Scrapy downloader.

Bu değeri artırıp azaltmak computer performance'la alakalı bir şey, yani şu degree set et demek doğru olmaz, ancak optimum performans için CPU Usage'ın %80-%90 arasında olması gerekiyor.

## #SOLUTION 2

➡ Add a delay between each request sent

➡ Prevent hitting the server too hard

➡ `DOWNLOAD_DELAY = 0`

Servera fazla yüklenmemek için, request'ler arası delay yerleştirebiliriz.

Bunu yapmak için settings.py içerisindeki `DOWNLOAD_DELAY` variable'ını kullanırız. Bu değer default olarak 0 seconds'a ayarlanmıştır. Bu değeri 5 yapmak istersek, settings.py'ye gireriz ilgili satırı uncomment edip değeri 5 olarak değiştiririz:

```
settings.py
17
18 # Crawl responsibly by identifying yourself (and your website) on the user-agent
19 # USER_AGENT = 'demo_crawl (+http://www.yourdomain.com)'
20
21 # Obey robots.txt rules
22 ROBOTSTXT_OBEY = True
23
24 # Configure maximum concurrent requests performed by Scrapy (default: 16)
25 #CONCURRENT_REQUESTS = 32
26
27 # Configure a delay for requests for the same website (default: 0)
28 # See https://doc.scrapy.org/en/latest/topics/settings.html#download-delay
29 # See also autothrottle settings and docs
30 DOWNLOAD_DELAY = 5
31
```

Bu ayarı yapınca, her requestten sonra tam 5 saniye delay olmaz onun yerine her requestten sonra random olarak  $0.5 \times 5$  saniye ile  $1.5 \times 5$  saniye arasında bir süre delay olur.

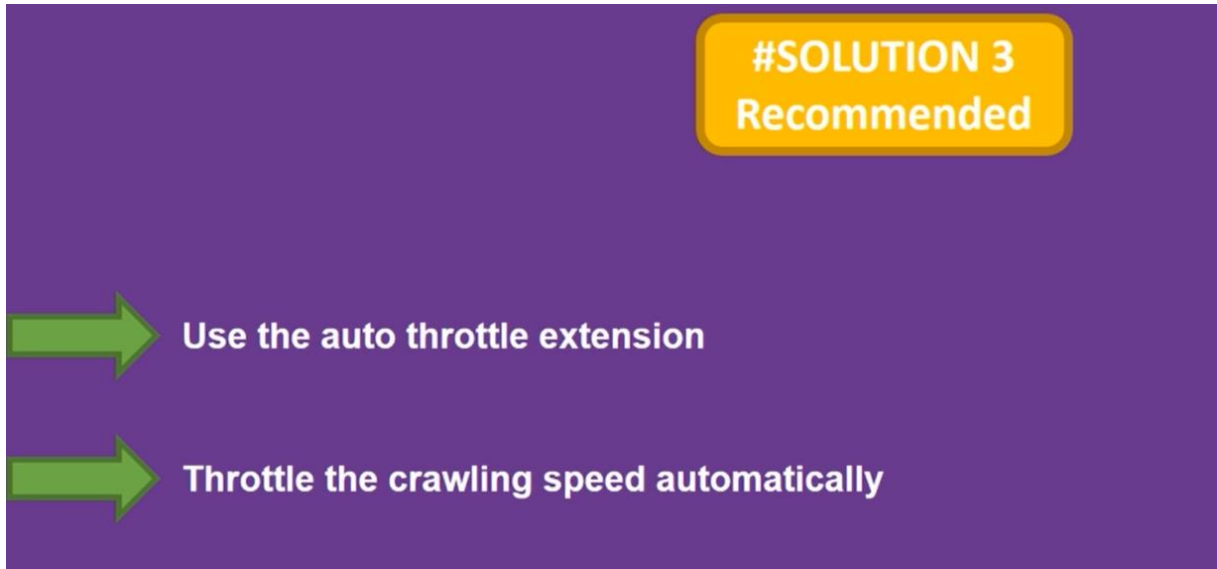
➡ `DOWNLOAD_DELAY = DOWNLOAD_DELAY * RANDOM_NUMBER[ 0.5 – 1.5 ]`

➡ `RANDOMIZE_DOWNLOAD_DELAY = False`

Eğer bu random'lığı istemezsek, her request sonrası 5 saniye beklesin istersek, `RANDOMIZE_DOWNLOAD_DELAY = False` satırını settings.py'ye eklemeliyiz.

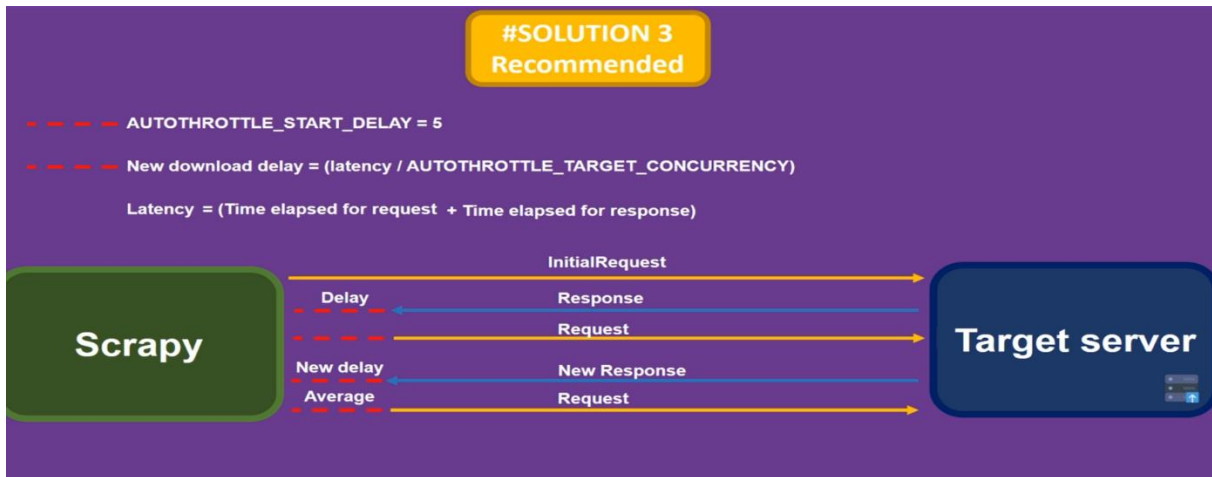
Ancak birden fazla spider'ın kullanıldığı büyük projelerde `DOWNLOAD_DELAY` variable'ını kullanarak requestler arası delay vermek işe yaramaz. Her spider farklı bir websitesini scrape etmekten sorumlu olabilir ve farklı serverlar farklı sürelerde delaylere ihtiyaç duyuyor olabilir.

İşte bu noktada diğer iki solution'ı da baskıyan ve bu durumu da gözeten son solution'ımız ortaya çıkıyor: AUTO THROTTLE EXTENSION.



Bu yöntem ile crawling speed otomatik olarak yavaşlatılacaktır.

Auto Throttle Extension'ın nasıl çalıştığını anlayalım:



- Servere initial request gönderiliyor, ve response alınıyor bu noktada AUTOTHROTTLE\_START\_DELAY variable'ı ile belirlenen bir delay spider tarafından devreye alınıyor. Bu delayden sonra response scrapy tarafından download ediliyor.
- Dolayısıyla 2. Request de bu süre kadar delayli gönderilmiş oluyor.
- Yeni bir response elde edildiğinde yeni delay yukarıdaki gibi hesaplanıyor.

Bu olayları tamamen anlamak gerekmiyor, bir fikir oluşsun diye verildi, uygulamasını görelim:

- Elbette bu yöntemi uygulayacaksak solution 2'de tanımlanan settings.py içerisindeki delay satırını commente alıyoruz:

```
# See also autothrrottle.py
DOWNLOAD_DELAY = 5
```

- Daha sonra yine settings.py içerisindeki **AUTOTHROTTLE\_ENABLE = True** satırını uncomment ediyoruz.


```
# Enable and configure the AutoThrottle extension (disabled by default)
# See https://doc.scrapy.org/en/latest/topics/autothrrottle.html
AUTOTHROTTLE_ENABLED = True
# The initial download delay
#AUTOTHROTTLE_START_DELAY = 5
# The maximum download delay to be set in case of high latencies
#AUTOTHROTTLE_MAX_DELAY = 60
# The average number of requests Scrapy should be sending in parallel to
# each remote server
#AUTOTHROTTLE_TARGET_CONCURRENCY = 1.0
# Enable showing throttling stats for every response received:
# AUTOTHROTTLE_DEBUG = True
```


- Burada diğer parametreleri de customize edebiliriz ancak default value'leri bozmaya gerek yok.




Siteye bir anda fazla yüklenmemek için kullanılabilecek son solution da **Enabling HTTP Caching**:

#SOLUTION 4

 **Enable HTTP caching**

 **Store the requests in the memory cache**

 **Use it only in the development stage(testing)**

- Sanıyorum bu yöntem test stage'inde kullanılıyor. Bir kez spider çalıştırıldığında requestleri ve response'ları kaydediyor.
- Daha sonra birşeyler değiştirip tekrar çalıştırsak, yeniden siteye request yollamak yerine cache edilmiş requestleri ve response'ları kullanıyor.
- http Caching'i sadece development stage için activate ederiz!

```
# Enable and configure HTTP caching (disabled by default)
# See https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#httpcache-middleware-settings
HTTPCACHE_ENABLED = True
HTTPCACHE_EXPIRATION_SECS = 0
HTTPCACHE_DIR = 'httpcache'
HTTPCACHE_IGNORE_HTTP_CODES = []
HTTPCACHE_STORAGE = 'scrapy.extensions.httpcache.FilesystemCacheStorage'
```

## RULE 2

Eğer robots.txt'ye uymak istersek ROBOTSTXT\_OBEY = True olarak bırakmalıyız, yok eğer robots.txt'ye uymak istemiyorsak bu variable'ı false olarak set etmeliyiz.



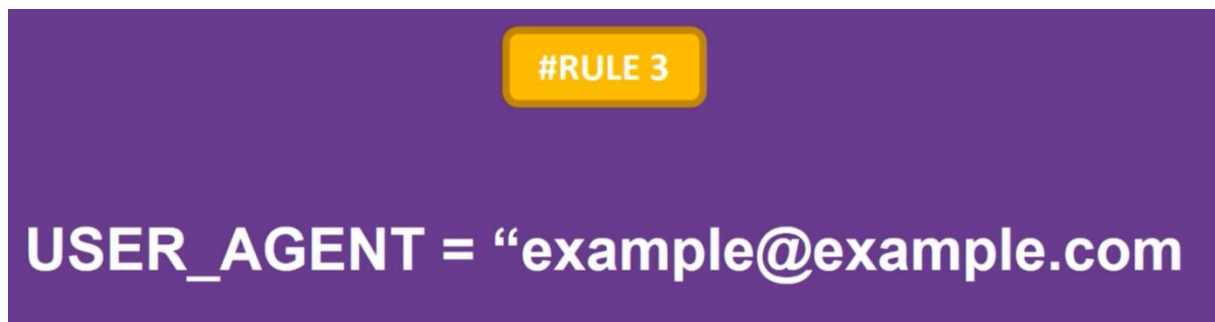
## RULE 3

User-Agent header'ı modifiye ederek websitesini kandır.

Bir sonraki başlıkta göreceğimiz, CUSTOM USER AGENT ROTATOR MIDDLEWARE kullanarak, request'ler farklı farklı user-agent'lardan geliyormuş gibi websitesini kandırırız.

Rule 3 aşağıdaki gibi tanımlanmıştı, ancak bu uygulanmaz onun yerine sen yukarıdakini rule olarak düşün.

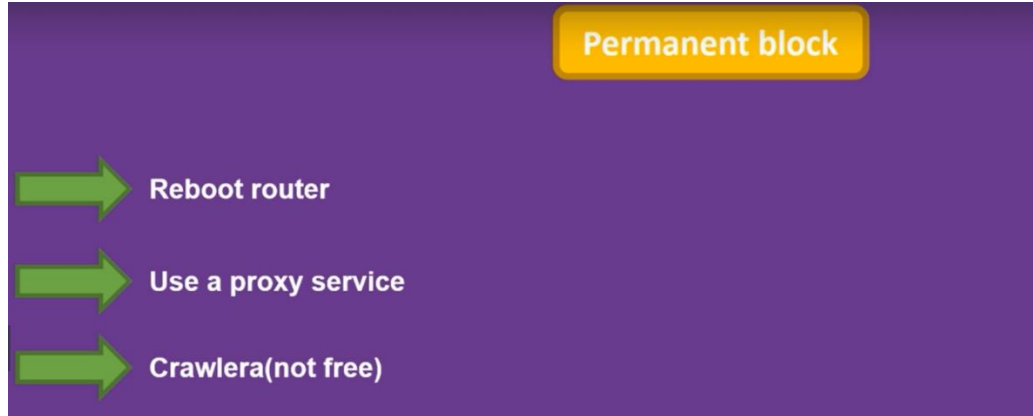
Genelde pek uygulanmayan bu kural ise şunu söyler, user agent header içerisinde bir contact email bulundurmak böylece eğer bir site crawler ile problem yaşarsa, bu adresi kullanarak bizimle iletişime geçebilir.



Ancak bunun yerine bir sonraki başlıkta göreceğimiz, CUSTOM USER AGENT ROTATOR MIDDLEWARE kullanarak, request'ler farklı farklı user-agent'lardan geliyormuş gibi websitesini kandırırız.

## What happens in case of permanent ban?

Yukarıda botumuzun banlanmasını engellemek için bazı yöntemler gördük, temporary IP banlar problem olmaz ama peki ya permanent ban yersek ne olacak?



- Modem'e reset atmak, IP adresi yenileyebileceği için permanent block yersek bu yöntemi kullanabiliriz. Ancak bu da internet sağlayıcımıza bağlı, duruma göre modemi resetlese de IP adresimiz değişmiyor olabilir.
- Eğer IP adresi modem resetleme ile yenilenmiyorsa paid Proxy service kullanmamız gerek.
- Crawlera bu amaçla kullanabilecek efficient bir Proxy.

## Sometimes we can not Scrape



Google recaptcha gibi bot tester'ları an itibari ile atlatmak mümkün değil, ancak yukarıda kurallara uyulduğu sürece, yani özetle çok hızlı crawling yapılmadığı ve doğru user-agent header'lar kullanıldığı sürece ve robot.txt'ye uyup uymayacağımız belirtildiği sürece, yakalanmadan crawling yapmamız yüksek olasılık.

## Avoid Getting Banned Part 3 – Building Custom User-Agent Rotator Middleware:

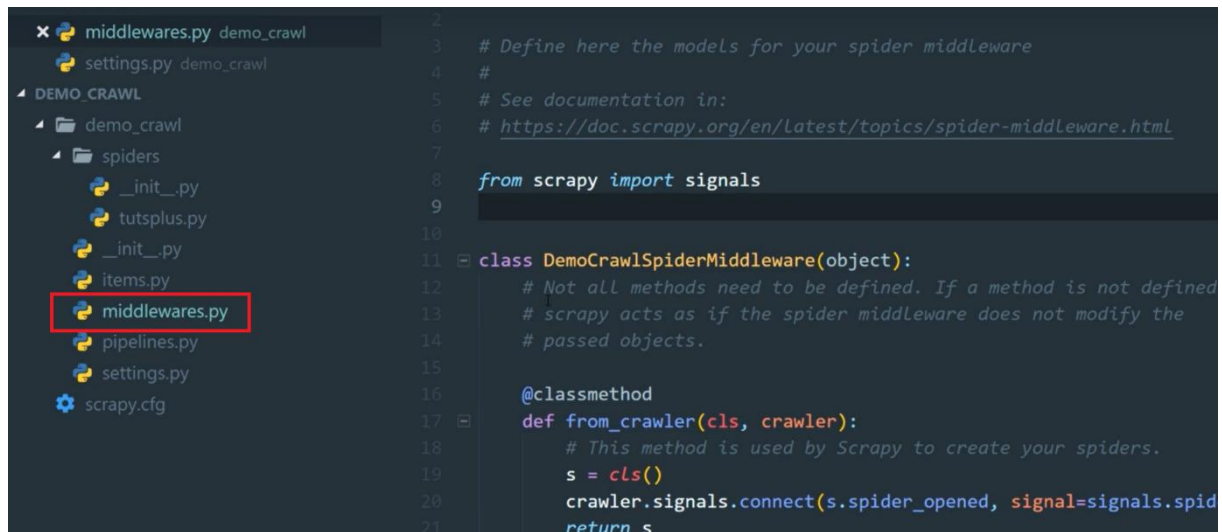
Bu videonun amacı önceki kısmın 3. Rule’unda bahsedilen işlemi gerçekleştirmek. Yani crawler’ın request gönderirken random olarak farklı farklı user-agent’lar kullanmasını sağlamak, böylece ban riskini düşürmek.

Random dediysek user-agent header’lar predefined bir listenin içinden random olarak seçiliyor, tamamen rastgele değerler atanmıyor.

Daha önce User-Agent header’ı değiştirmeyi görmüştük ancak daha önce bunu yaptığımızda tüm requestlerde aynı User-Agent header kullanılmıştı şimdi ise bu değer bir dictionary içerisinde rastgele olarak seçilmesini sağlayacağız.

Uygulamaya geçelim:

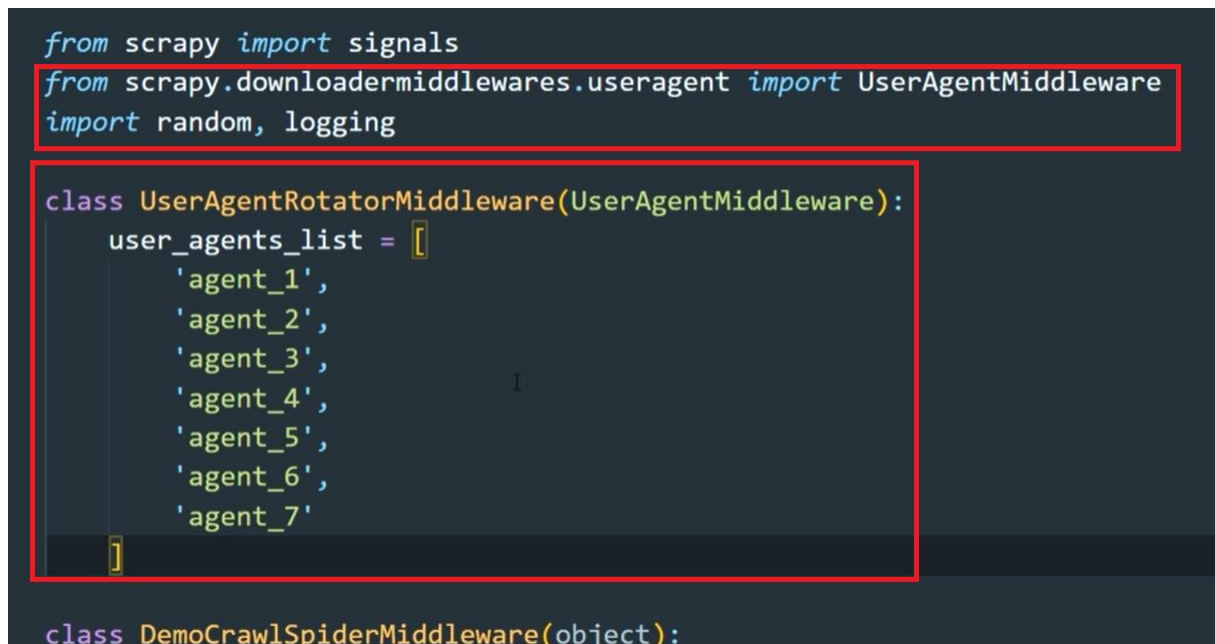
→Öncelikle proje içerisindeki middlewares.py dosyasına giriyoruz.



```
1 # Define here the models for your spider middleware
2 #
3 # See documentation in:
4 # https://doc.scrapy.org/en/latest/topics/spider-middleware.html
5
6 from scrapy import signals
7
8
9
10
11 class DemoCrawlSpiderMiddleware(object):
12     # Not all methods need to be defined. If a method is not defined
13     # scrapy acts as if the spider middleware does not modify the
14     # passed objects.
15
16     @classmethod
17     def from_crawler(cls, crawler):
18         # This method is used by Scrapy to create your spiders.
19         s = cls()
20         crawler.signals.connect(s.spider_opened, signal=signals.spider_opened)
21         return s
```

→Gerekli module'leri import ettikten sonra aşağıdaki gibi bir subclass tanımlayıp içerisine kullanmak istediğim user-agent header'ları bir list olarak veriyorum.

→Aşağıda verilen header'lar örnek için verilmiş gerçekte bunların hepsinin gerçek header'lar olması gerek, böyle listelere google'dan ulaşabilirim "User-Agent Header Lists" falan şeklinde. Ya da kursta bu videoya eklenmiş şekilde user-agent header list de bulunuyor.



```
from scrapy import signals
from scrapy.downloadermiddlewares.useragent import UserAgentMiddleware
import random, logging

class UserAgentRotatorMiddleware(UserAgentMiddleware):
    user_agents_list = [
        'agent_1',
        'agent_2',
        'agent_3',
        'agent_4',
        'agent_5',
        'agent_6',
        'agent_7'
    ]

class DemoCrawlSpiderMiddleware(object):
```

MiddleWare'ı modify etmeye devam ediyoruz:

```
class UserAgentRotatorMiddleware(UserAgentMiddleware):
    user_agents_list = [
        'agent_1',
        'agent_2',
        'agent_3',
        'agent_4',
        'agent_5',
        'agent_6',
        'agent_7'
    ]

    def __init__(self, user_agent=''):
        self.user_agent = user_agent

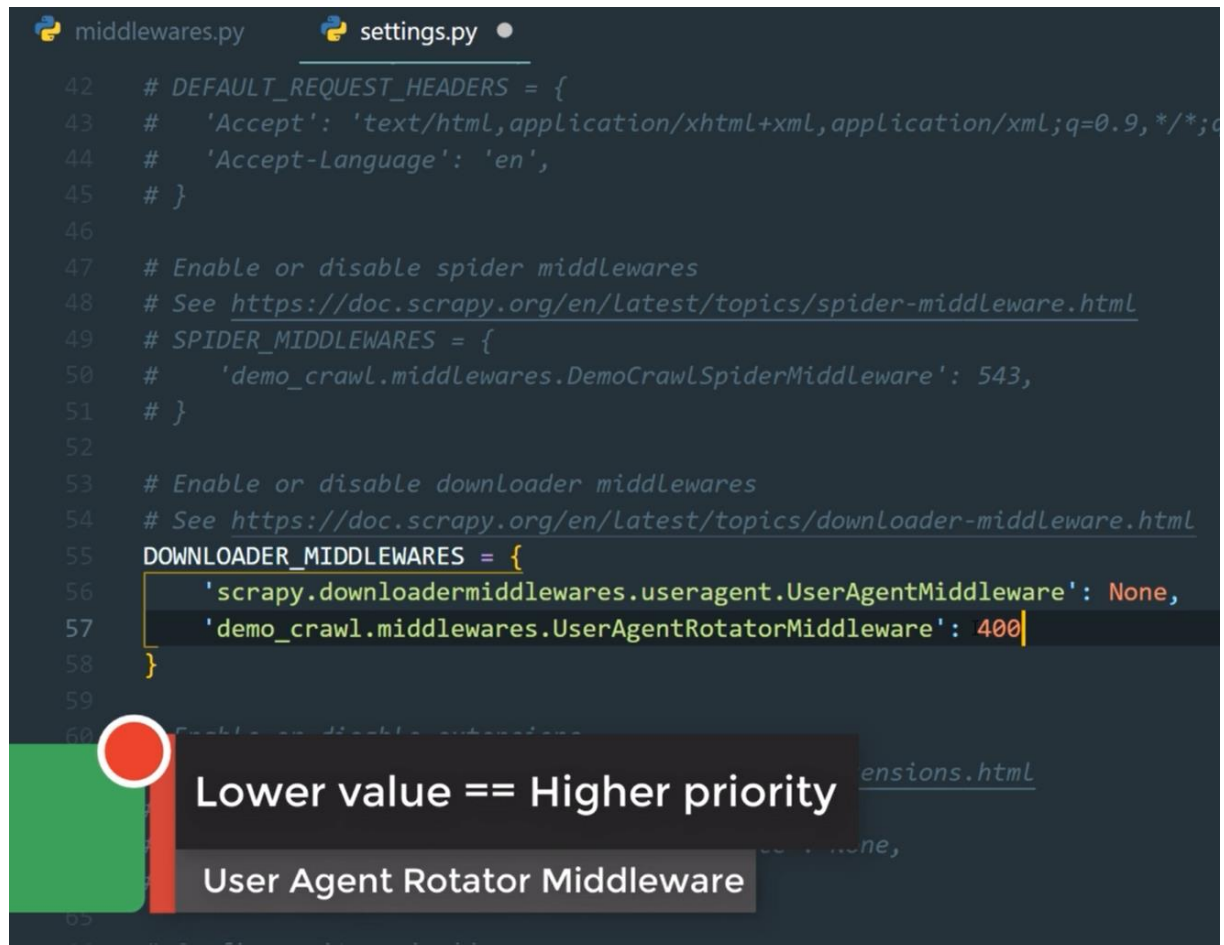
    def process_request(self, request, spider):
        try:
            self.user_agent = random.choice(self.user_agents_list)
            request.headers.setdefault('User-Agent', self.user_agent)
        except IndexError:
            logging.error("Couldn't fetch the user agent")

class DemoCrawlSpiderMiddleware(object):
```

- User\_agents\_list'den sonra bir constructor method tanımlanıyor.
- Daha sonra process\_request isimdeki methodu yukarıdaki gibi override ediyoruz.

Böylelikle middleware.py methodu ile işimiz tamam.

Şimdi spider'ımızı crawling sırasında oluşturulan middleware'ı kullanması için settings.py içerisinde aşağıdaki değişiklikleri yapıyoruz.



The screenshot shows a code editor with two tabs: 'middlewares.py' and 'settings.py'. The 'settings.py' tab is active, displaying Python code for Scrapy settings. The code includes comments for default request headers, enabling/disabling spider middlewares, and enabling/disabling downloader middlewares. The 'DOWNLOADER\_MIDDLEWARES' dictionary is defined, with 'scrapy.downloadermiddlewares.useragent.UserAgentMiddleware' set to 'None' and 'demo\_crawl.middlewares.UserAgentRotatorMiddleware' set to '400'. A callout box with a red circle and a green bar contains the text: 'Lower value == Higher priority' and 'User Agent Rotator Middleware'.

```
42 # DEFAULT_REQUEST_HEADERS = {
43 #     'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8',
44 #     'Accept-Language': 'en',
45 # }
46
47 # Enable or disable spider middlewares
48 # See https://doc.scrapy.org/en/latest/topics/spider-middleware.html
49 # SPIDER_MIDDLEWARES = {
50 #     'demo_crawl.middlewares.DemoCrawlSpiderMiddleware': 543,
51 # }
52
53 # Enable or disable downloader middlewares
54 # See https://doc.scrapy.org/en/latest/topics/downloader-middleware.html
55 DOWNLOADER_MIDDLEWARES = {
56     'scrapy.downloadermiddlewares.useragent.UserAgentMiddleware': None,
57     'demo_crawl.middlewares.UserAgentRotatorMiddleware': 400
58 }
59
60 # Enable or disable extensions
61 # See https://doc.scrapy.org/en/latest/topics/extensions.html
62 # EXTENSIONS = {
63 #     'scrapy.extensions.telnet.TelnetConsole': None,
64 # }
```

- Özetle DOWNLOADER\_MIDDLEWARES'i uncomment edip içeriğini yukarıdaki gibi set ediyoruz.

Artık spider'ımız crawling sırasında her requesti random olarak farklı bir user-agent header ile gönderecek.

Böylece siteden ban yeme ihtimalimiz düşecektir !!!

Çalışıp çalışmadığını test etmek için spider'ın yield ettiği değerlerin yanına bir de user-agent variable eklenebilir böylece outputa bakıp gerçekten farklı farklı user-agent header'ların kullanıldığından emin olabiliriz:

```
tutorial_id = response.xpath("//a[@class='posts__post-title ']/h1/text()").extract_first(),
yield {
    'title': tutorial.xpath("//a[@class='posts__post-title ']/h1/text()").extract_first(),
    'url': tutorial.xpath("//a[@class='posts__post-title ']/@href").extract_first(),
    'category': response.xpath("//span[@class='content-banner title-breadcrumb-category']/text()").extract_first(),
    'user-agent': response.request.headers.get('User-Agent').decode('utf-8')
}
```