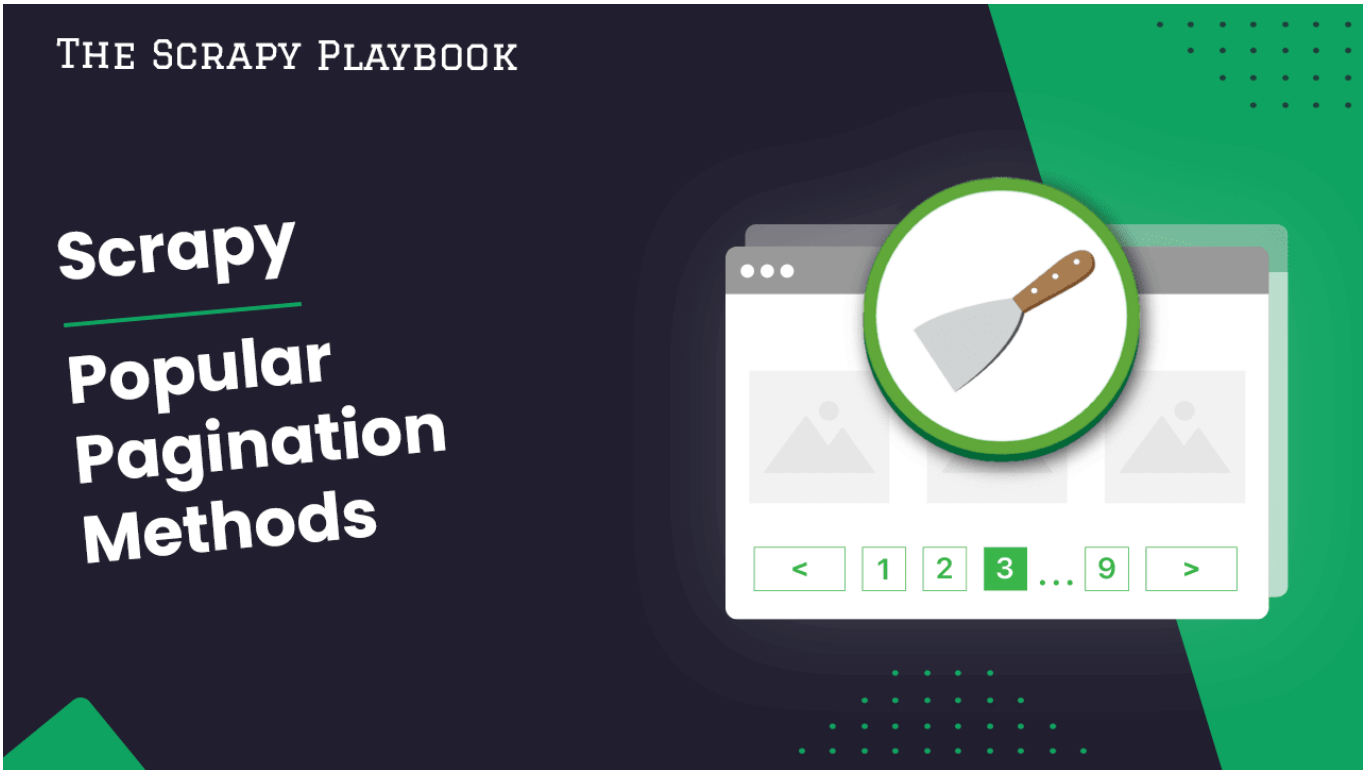

Scrapy Pagination Guide: The 6 Most Popular Pagination Methods

To scrape at scale, you need to be able to deal with whatever pagination system the website throws at you.

In this guide, we're going to walk through **6 of the most common pagination methods** you can use to scape the data you need:

1. Change Page Number In URL
2. Follow Next Page URL From Response
3. Using a Websites Sitemap
4. Using CrawlSpider
5. Paginate API Requests
6. Use Machine Learning With Autopager

#1: Change Page Number In URL

The simplest pagination type you will see is when the website site changes pages by just changing a page number in the URL.

A good example of this is the quotes.toscrape.com website, where it just uses page numbers for pagination:

```
https://quotes.toscrape.com/
https://quotes.toscrape.com/page/2/
https://quotes.toscrape.com/page/3/
```

Here, we can just write a simple script to loop through page numbers and:

- Stops because we've defined a fixed depth. Either because we know the last page number, or only want to go X pages deep.
- Stops when we get a `404` response as the page number we are requesting doesn't exist.

Both of **these options aren't the Scrapy way of solving pagination**, but they work. Here is how you can use either approach.

Stop at a Fixed Page Depth

When you either know the maximum number of pages, or if you only want to scrape a fixed number of pages you can use this approach.

In this example, we're going to pass `start_urls` with a list of urls with page numbers from 1 to 10 as there are only 10 pages available on the site.

```
import scrapy
from pagination_demo.items import QuoteItem

class QuotesSpider(scrapy.Spider):
    name = "quotes_fixed"
    start_urls = ["http://quotes.toscrape.com/page/%d/" % i for i in range(1,11)]

    def parse(self, response):
        quote_item = QuoteItem()
        for quote in response.css('div.quote'):
            quote_item['text'] = quote.css('span.text::text').get()
            quote_item['author'] = quote.css('small.author::text').get()
            quote_item['tags'] = quote.css('div.tags a.tag::text').getall()
            yield quote_item
```

It's simple and works, but requires you to know how many pages there will be.

Stop When We Get 404 Status Code Or Data Is Missing

The other way of paginating through a site like this is to start at **page number 1**, and stop when we get a `404` response or for quotes.toscrape.com stop when we request a page with no quotes on it (it doesn't give `404` responses).

```
import scrapy
from pagination_demo.items import QuoteItem
from scrapy.exceptions import CloseSpider


class QuotesSpider(scrapy.Spider):
    name = "quotes_404"
    start_urls = ["http://quotes.toscrape.com/page/1/"]
    handle_httpstatus_list = [404] # to catch 404 with callback
    page_number = 1

    def parse(self, response):

        # stop spider on 404 response
        if response.status == 404:
            raise CloseSpider('Recieve 404 response')

        # stop spider when no quotes found in response
        if len(response.css('div.quote')) == 0:
            raise CloseSpider('No quotes in response')

        quote_item = QuoteItem()
        for quote in response.css('div.quote'):
            quote_item['text'] = quote.css('span.text::text').get()
            quote_item['author'] = quote.css('small.author::text').get()
            quote_item['tags'] = quote.css('div.tags a.tag::text').getall()
            yield quote_item

        # go to next page
        self.page_number += 1
        next_page = f'http://quotes.toscrape.com/page/{self.page_number}/'
        yield response.follow(next_page, callback=self.parse)
```

Now that you have seen two non-Scrapy ways to approaching pagination, next we will show the Scrapy way.

#2: Follow Next Page URL From Response

The Scrapy way of solving pagination would be to use the url often contained in **next page** button to request the next page.

Again, when looking at quotes.toscrape.com, we need to extra the URL from the **Next** button at the bottom of the page and use it in the next request.

Here our scraper extracts the relative URL from the **Next** button:

```
In [2]: response.css("li.next a::attr(href)").extract_first()
Out[2]: '/page/2/'
```

Which then gets joined to the base url by the `response.follow(next_page, callback=self.parse)` and makes the request for the next page. This process keeps going until the `next_page` is `None`:

```
import scrapy
from pagination_demo.items import QuoteItem


class QuotesSpider(scrapy.Spider):
    name = "quotes_button"
    start_urls = ["http://quotes.toscrape.com/"]

    def parse(self, response):
        quote_item = QuoteItem()
        for quote in response.css('div.quote'):
            quote_item['text'] = quote.css('span.text::text').get()
            quote_item['author'] = quote.css('small.author::text').get()
            quote_item['tags'] = quote.css('div.tags a.tag::text').getall()
            yield quote_item

        # go to next page
        next_page = response.css("li.next a::attr(href)").extract_first()
        if next_page:
            yield response.follow(next_page, callback=self.parse)
```

This method is more versatile and will work in simple situations where the website paginates just with page numbers or in more complex situations where the website uses more complicated query parameters.

#3: Using a Websites Sitemap

Sometimes if a website is heavily optimising itself for SEO, then using their own sitemap is a great way to remove the need for pagination altogether.

Here we can use Scrapy's SitemapSpider, to extract the URLs that match our criteria from their sitemap and then have Scrapy scrape them as normal.

Oftentimes, a websites sitemap is located at `https://www.demo.com/sitemap.xml` so you can quickly check if the site has a sitemap, and if it contains the URLs you are looking for.

Quotes.toscrape.com doesn't have a sitemap, so for this example we will scrape all the article URLs and titles from ScraperAPI's blog using their sitemap.

```
from scrapy.spiders import SitemapSpider
from pagination_demo.items import BlogItem


class BlogSpider(SitemapSpider):
    name = "scraperapi_sitemap"
    sitemap_urls = ['https://www.scraperapi.com/post-sitemap.xml']
    sitemap_rule = [
     ('blog/', 'parse'),
   ]

    def parse(self, response):
        blog_item = BlogItem()
```

```
        blog_item['url'] = response.url
        blog_item['title'] = response.css('h1 > span::text').get()
        yield blog_item
```

#4: Using CrawlSpider

While not exactly pagination, in situations you would like to scrape all pages of a specific type you can use a CrawlSpider and leave it find and scrape the pages for you.

When using **CrawlSpider** you will need to specify the `allowed_domains` and the crawling rules so that it will only scrape the pages you want to scrape.

In the quotes.toscrape.com example below, we specify that we only want it to scrape pages that include `page/` in the URL, but exclude `tag/`.

As otherwise we would be scraping the tag pages too as they contain `page/` as well `https://quotes.toscrape.com/tag/heartbreak/page/1/`.

```
import scrapy
from scrapy.spiders import CrawlSpider, Rule
from scrapy.linkextractors import LinkExtractor
from pagination_demo.items import QuoteItem


class QuotesSpider(CrawlSpider):
    name = "quotes_crawler"
    allowed_domains = ["quotes.toscrape.com"]
    start_urls = ["http://quotes.toscrape.com/"]

    rules = (
        Rule(LinkExtractor(allow = 'page/', deny='tag/'), callback='parse', follow=True),
    )

    def parse(self, response):
        quote_item = QuoteItem()
        for quote in response.css('div.quote'):
            quote_item['text'] = quote.css('span.text::text').get()
            quote_item['author'] = quote.css('small.author::text').get()
            quote_item['tags'] = quote.css('div.tags a.tag::text').getall()
            yield quote_item
```

Using the **CrawlSpider** approach is good as you can let it find pages that match your criteria. However, in can be an inefficent approach as it could scrape more pages than is necessary and it might miss some pages.

If you know the exact pages you would like to scrape and can figure out the pagination method the website uses, then it is normally better just to reverse engineer that.

#5: Paginate API Requests

If we are scraping an API oftentimes, it will be paginated and only return a set number of results per response. This is normally a pretty easy problem to solve.

Looking at The Rick and Morty API as an example, we can see that in every response it returns the url of the next page.

```
## GET https://rickandmortyapi.com/api/character/

{
    "info": {
        "count": 826,
        "pages": 42,
        "next": "https://rickandmortyapi.com/api/character/?page=2",
        "prev": null
    },
    "results":[...],
}
```

Follow The Next URL

One option is extract this url and have Scrapy request it with `response.follow()`.

```
import scrapy
from pagination_demo.items import CharacterItem

class CharacterSpider(scrapy.Spider):
    name = "api_pagination"
    start_urls = ["https://rickandmortyapi.com/api/character/"]

    def parse(self, response):
        json_response = response.json()
        character_item = CharacterItem()
        for character in json_response.get('results', []):
            character_item['name'] = character.get('name')
            character_item['status'] = character.get('status')
            character_item['species'] = character.get('species')
            yield character_item

        next_page = json_response['info'].get('next')
        if next_page:
            yield response.follow(next_page, callback=self.parse)
```

Generate Requests For Every Page

Since the response also includes the total number of pages `"pages": 42`, and we can see from the URL that it is just paginating using a `?page=2` query parameter, we can have our spider generate all the requests after the first response.

```
import scrapy
from pagination_demo.items import CharacterItem

class CharacterSpider(scrapy.Spider):
    name = "api_pagination"
    start_urls = ["https://rickandmortyapi.com/api/character/"]

    def parse(self, response):
        json_response = response.json()
        character_item = CharacterItem()
        for character in json_response.get('results', []):
            character_item['name'] = character.get('name')
            character_item['status'] = character.get('status')
            character_item['species'] = character.get('species')
            yield character_item

        num_pages = json_response['info'].get('pages')
        for page in range(2, num_pages + 1):
            next_page = f'https://rickandmortyapi.com/api/character/?page={page}'
            yield response.follow(next_page, callback=self.parse)
```

This option is a faster method to extract all the data than the first option, as it will send all the URLs to the Scrapy scheduler at the start and have them processed in parallel. Instead, of processing the pages one after the other as will happen with the first approach.

#6: Use Machine Learning With Autopager

Autopager is a Python package that detects and classifies pagination links on a page, using a pre-trained machine learning model.

Using Autopager, you can have it detect what pagination schema a website is using and then integrate this into your Scrapy spider using one of the above approaches.

To use Autopager, first install the Python package:

```
pip install autopager
```

Then give it an example page for it to detect the pagination schema:

```
>>> import autopager
>>> import requests
>>> autopager.urls(requests.get('http://quotes.toscrape.com'))
```

```
['http://quotes.toscrape.com/tag/obvious/page/1/', 'http://quotes.toscrape.com/tag/simile/page/1/', 'http://
quotes.toscrape.com/page/2/']
```

You could try and directly integrte Autopager into your Scrapy spider, however, as it only extracts the pagination links it finds on the example page you provide it won't generate all the pagination links you need. Plus, it can catch pagination links that you mightn't be interested in.

So the best way to use it is when you are first investigating a page, you provide an example URL and use it to quickly identify the pagination mechanism.

The team behind Autopager, say it should detect the pagination mechanism in **9/10** websites.

More Scrapy Tutorials

That's it for all the pagination techniques we can use with Scrapy. If you know of more then let us know in the comments section below.

If you would like to learn more about Scrapy, then be sure to check out The Scrapy Playbook.