

# 기초실습

06.13

최성은

### 크롤링의 이해 및 기본

#### 크롤링 (Crawling)

- Web상에 존재하는 Contents를 수집하는 작업 (프로그래밍으로 자동화 가능)
- HTML 페이지를 가져와서, HTML/CSS등을 파싱하고, 필요한 데이터만 추출하는 기법
- Open API(Rest API)를 제공하는 서비스에 Open API를 호출해서, 받은 데이터 중 필요한 데이터만 추출하는 기법
- Selenium등 브라우저를 프로그래밍으로 조작해서, 필요한 데이터만 추출하는 기법

```
import requests
from bs4 import BeautifulSoup

# 1) requests 라이브러리를 활용한 HTML 페이지 요청
# 1-1) res 객체에 HTML 데이터가 저장되고, res.content로 데이터를 추출할 수 있음
res = requests.get('http://v.media.daum.net/v/20170615203441266')

# print(res.content)
# 2) HTML 페이지 파싱 BeautifulSoup(HTML 데이터, 파싱방법)
# 2-1) BeautifulSoup 파싱방법
soup = BeautifulSoup(res.content, 'html.parser')

# 3) 필요한 데이터 검색
title = soup.find('title')

# 4) 필요한 데이터 추출
print(title.get_text())
```

잔금대출에도 DTI 규제 적용 검토

#### BeautifulSoup 라이브러리

- HTML의 태그를 파싱해서
- 필요한 데이터만 추출하는 함수를 제공하는 라이브러리

### 크롤링의 이해 및 기본

#### BeautifulSoup 라이브러리

- find() 와 find\_all() 메서드 사용법 이해하기
- find() : 가장 먼저 검색되는 태그 반환
- find\_all() : 전체 태그 반환

```
from bs4 import BeautifulSoup

html = """
<html>
  <body>
    <h1 id='title'>[1] 크롤링이란?</h1>
    <p class='cssstyle'>웹페이지에서 필요한 데이터를 추출하는 것</p>
    <p id='body' align='center'>파이썬을 중심으로 다양한 웹크롤링 기술 발달</p>
  </body>
</html>
"""

soup = BeautifulSoup(html, "html.parser")

# 태그로 검색 방법
title_data = soup.find('h1')

print(title_data)
print(title_data.string)
print(title_data.get_text())

<h1 id="title">[1] 크롤링이란?</h1>
[1] 크롤링이란?
[1] 크롤링이란?
```

```
# 태그에 있는 id로 검색 (javascript 예를 상기!)
title_data = soup.find(id='title')

print(title_data)
print(title_data.string)
print(title_data.get_text())
```

```
<h1 id="title">[1] 크롤링이란?</h1>
[1] 크롤링이란?
[1] 크롤링이란?
```

```
# HTML 태그와 CSS class를 활용해서 필요한 데이터를 추출하는 방법1
paragraph_data = soup.find('p', class_='cssstyle')

print(paragraph_data)
print(paragraph_data.string)
print(paragraph_data.get_text())
```

```
<p class="cssstyle">웹페이지에서 필요한 데이터를 추출하는 것</p>
웹페이지에서 필요한 데이터를 추출하는 것
웹페이지에서 필요한 데이터를 추출하는 것
```

```
# HTML 태그와 CSS class를 활용해서 필요한 데이터를 추출하는 방법2
paragraph_data = soup.find('p', 'cssstyle')

print(paragraph_data)
print(paragraph_data.string)
print(paragraph_data.get_text())
```

```
<p class="cssstyle">웹페이지에서 필요한 데이터를 추출하는 것</p>
웹페이지에서 필요한 데이터를 추출하는 것
웹페이지에서 필요한 데이터를 추출하는 것
```

### 크롤링의 이해 및 기본

#### BeautifulSoup 라이브러리

- find() 와 find\_all() 메서드 사용법 이해하기
- find() : 가장 먼저 검색되는 태그 반환
- find\_all() : 전체 태그 반환

```
from bs4 import BeautifulSoup

html = """
<html>
  <body>
    <h1 id='title'>[1] 크롤링이란?</h1>
    <p class='cssstyle'>웹페이지에서 필요한 데이터를 추출하는 것</p>
    <p id='body' align='center'>파이썬을 중심으로 다양한 웹크롤링 기술 발달</p>
  </body>
</html>
"""

soup = BeautifulSoup(html, "html.parser")

# 태그로 검색 방법
title_data = soup.find('h1')

print(title_data)
print(title_data.string)
print(title_data.get_text())
```

```
<h1 id="title">[1] 크롤링이란?</h1>
[1] 크롤링이란?
[1] 크롤링이란?
```

```
# HTML 태그와 태그에 있는 속성:속성값을 활용해서 필요한 데이터를 추출하는 방법
paragraph_data = soup.find('p', attrs = {'align': 'center'})
print(paragraph_data)
print(paragraph_data.string)
print(paragraph_data.get_text())
```

```
<p align="center" id="body">파이썬을 중심으로 다양한 웹크롤링 기술 발달</p>
파이썬을 중심으로 다양한 웹크롤링 기술 발달
파이썬을 중심으로 다양한 웹크롤링 기술 발달
```

```
# find_all() 관련된 모든 데이터를 리스트 형태로 추출하는 함수
paragraph_data = soup.find_all('p')

print(paragraph_data)
print(paragraph_data[0].get_text())
print(paragraph_data[1].get_text())
```

```
[<p class="cssstyle">웹페이지에서 필요한 데이터를 추출하는 것</p>,
<p align="center" id="body">파이썬을 중심으로 다양한 웹크롤링 기술 발달</p>]
웹페이지에서 필요한 데이터를 추출하는 것
파이썬을 중심으로 다양한 웹크롤링 기술 발달
```

# Chapter 4.

# Web crawling model

06.13

최성은

### 4.2 다양한 웹사이트 레이아웃 다루기

- 가장 확실한 방편은 각 웹사이트에 대해 별도의 웹 크롤러 또는 페이지 구문 분석기를 만드는 것
- 각각은 URL, 문자열 또는 BeautifulSoup 객체를 받아, 스크랩한 내용을 파이썬 객체로 반환

### Dealing with different website layouts

```
import requests
```

Content 클래스

```
class Content:
```

```
    def __init__(self, url, title, body):
        self.url = url
        self.title = title
        self.body = body
```

```
def getPage(url):
```

```
    req = requests.get(url)
    return BeautifulSoup(req.text, 'html.parser')
```

```
def scrapeNYTimes(url):
```

```
    bs = getPage(url)
    title = bs.find('h1').text
    lines = bs.select('div.StoryBodyCompanionColumn div p')
    body = '\n'.join([line.text for line in lines])
    return Content(url, title, body)
```

```
def scrapeBrookings(url):
```

```
    bs = getPage(url)
    title = bs.find('h1').text
    body = bs.find('div', {'class', 'post-body'}).text
    return Content(url, title, body)
```

Content 인스턴스 반환

```
url = 'https://www.brookings.edu/blog/future-development/2018/01/26/delivering-inclusive-urban-access-3-uncomfortable-truths/'
content = scrapeBrookings(url)
print('Title: {}'.format(content.title))
print('URL: {}\n'.format(content.url))
print(content.body)
```

```
url = 'https://www.nytimes.com/2018/01/25/opinion/sunday/silicon-valley-immortality.html'
content = scrapeNYTimes(url)
print('Title: {}'.format(content.title))
print('URL: {}\n'.format(content.url))
print(content.body)
```

Title: Delivering inclusive urban access: 3 uncomfortable truths  
URL: <https://www.brookings.edu/blog/future-development/2018/01/26/delivering-inclusive-urban-access-3-uncomfortable-truths/>

The past few decades have been filled with a deep optimism about the role of cities. Economic growth, the source of jobs and innovation, has been the engine of economic growth. The past few decades have been filled with a deep optimism about the role of cities. Economic growth, the source of jobs and innovation, has been the engine of economic growth. The past few decades have been filled with a deep optimism about the role of cities. Economic growth, the source of jobs and innovation, has been the engine of economic growth.

Title: The Men Who Want to Live Forever  
URL: <https://www.nytimes.com/2018/01/25/opinion/sunday/silicon-valley-immortality.html>








Would you like to live forever? Some billionaires, already invincible in every other way, would like to live forever. Today several biotech companies, fueled by Silicon Valley fortunes, are devoted to solving "the problem of death."


- 제목 추출
- 주요 콘텐츠 선택
- 콘텐츠 항목 선택
- 문자열로 Content 객체 반환
- 여기서 사용된 변수들 중 달라지는 변수는 CSS 선택자 뿐

### 4.2 다양한 웹사이트 레이아웃 다루기


### : 3 uncomfortable truths


urbanization, rising gentrification, concentrated poverty, and shortages of basic infrastructure have combined to create spatial inequity in cities and suburbs across the globe. The challenges of housing, moving, and employing so many people have led to longer travel times, rising housing costs, and unsustainable public spending. Moreover, policymakers are questioning traditional policies and approaches.





Economy and Development



**Adie Tomer**  
Fellow - Metropolitan Policy Program  
 AdieTomer

```
><div class="inner" style="width:150px;">...</div>
</div>
><a href="#" class="share-button share-more">...</a>
</li>
<li class="share-end"></li>
</ul>
</div>
</div>
</aside>
<div class="blog-content content-c">
  ><div class="language-select">...</div>
  ...
  ><div class="post-body post-body-articleBody"> == $0
html body div section div div.post-body.pc
post-body| 1 of
Styles Event Listeners DOM Breakpoints Pr
```

### 4.2 다양한 웹사이트 레이아웃 다루기

Today several biotech companies, fueled by Silicon Valley fortunes, are devoted to “life extension” — or as some put it, to solving “the problem of death.”

It’s a cause championed by the tech billionaire Peter Thiel, the TED Talk darling Aubrey de Gray, Google’s billion-dollar Calico longevity lab and investment by Amazon’s Jeff Bezos. The National Academy of Medicine, an independent group, recently dedicated funding to “end aging forever.”

As the longevity entrepreneur Arram Sabeti told The New Yorker: “The proposition that we can live forever is obvious. It doesn’t violate the laws of physics, so we can achieve it.” Of all the slightly creepy aspects to this trend, the strangest is the least noticed: The people publicly championing life extension are mainly men.

Not all of them, of course. In 2009, Elizabeth Blackburn received

```
<div id="main">
  <main id="site-content">
    <div>
      <div class="css-1aor85t" style="opacity: 1; visibility: visible;">
        <article id="story" class="css-1v...
          e1qksbhf0">
            <div id="NYT_TOP_BANNER_REGION"
              13pd83m">...</div>
            <div id="top-wrapper" class="css-...
              </div>
            <header class="css-6gfzut e12qa4...
              </header>
            <section name="articleBody" item...
              "articleBody" class="meteredConte...
                1r7ky0e">
                  <div class="css-1fanzo5...
                    StoryBodyCompanionColumn">...</div>
                  <div id="story-ad-1-wrapper" c...
```

html.story body

StoryBody

1 of 3

Styles

Event Listeners

DOM Breakpoints

Properties

Filter

:hov .cls

+

element.style {

padding-top: 0px;

margin

border



### 4.2 다양한 웹사이트 레이아웃 다루기

- 여기서 사용된 변수들 중 달라지는 변수는 CSS 선택자 뿐
- BeautifulSoup 의 find, find\_all 함수는 매개변수로 태그 문자열과 키/값 속성으로 이루어진 딕셔너리 받음
- 사이트 구조와 데이터 위치를 정의하는 매개변수를 넘기면 됨
- 수집할 정보의 CSS 선택자 문자열 하나로 만들고 딕셔너리 객체에 모아 BeautifulSoup의 select 함수 함께 사용하면 편리

#### 글/페이지 전체에 사용할 기반 클래스

```
class Content:
    """
    Common base class for all articles/pages
    """
    def __init__(self, url, title, body):
        self.url = url
        self.title = title
        self.body = body

    def print(self):
        """
        Flexible printing function controls output
        """
        print('URL: {}'.format(self.url))
        print('TITLE: {}'.format(self.title))
        print('BODY: \n{}'.format(self.body))
```

#### 웹사이트 구조에 관한 정보를 저장할 클래스

```
class Website:
    """
    Contains information about website structure
    """
    def __init__(self, name, url, titleTag, bodyTag):
        self.name = name
        self.url = url
        self.titleTag = titleTag
        self.bodyTag = bodyTag
```

- Website class
- 해당 데이터를 수집할 방법에 대한 지침을 저장
- Ex. titleTag, 제목을 찾을 수 있는 위치를 나타내는 문자열 태그, h1 저장

- Content, Website 클래스 사용하면 주어진 웹 페이지에 존재하는 URL의 제목과 내용을 모두 스크랩할 Crawler를 작성할 수 있음

### 4.2 다양한 웹사이트 레이아웃 다루기

- 사이트 구조와 데이터 위치를 정의하는 매개변수를 넘기면 됨
- 수집할 정보의 CSS 선택자 문자열 하나로 만들고 딕셔너리 객체에 모아 BeautifulSoup의 select 함수 함께 사용하면 편리

#### Class Content:

글/페이지 전체에 사용할 기반 클래스

#### Class Website:

웹사이트 구조에 관한 정보를 저장할 클래스

데이터 수집할 방법에 대한 지침

위치를 나타내는 태그 저장

#### Class Crawler:

Content, Website 클래스 사용하면 주어진 웹 페이지에 존재하는

URL 의 제목과 내용을 모두 스크랩할 Crawler 를 작성할 수 있음

#### Def getPage:

- Html 가져와서 html 태그로 파싱

#### Def safeGet:

- BeautifulSoup 객체와 선택자를 받아 콘텐츠 문자열을 추출하는 함수, 주어진 선택자로 검색된 결과가 없다면 빈 문자열을 반환함

#### Def parse:

- URL 을 받아 콘텐츠를 추출

```
class Crawler:
```

```
def getPage(self, url):
    try:
        req = requests.get(url)
    except requests.exceptions.RequestException:
        return None
    return BeautifulSoup(req.text, 'html.parser')
```

```
def safeGet(self, pageObj, selector):
    """
    Utility function used to get a content string from a Beautiful Soup
    object and a selector. Returns an empty string if no object
    is found for the given selector
    """
    selectedElems = pageObj.select(selector)
    if selectedElems is not None and len(selectedElems) > 0:
        return '\n'.join([elem.get_text() for elem in selectedElems])
    return ''
```

```
def parse(self, site, url):
    """
    Extract content from a given page URL
    """
    bs = self.getPage(url)
    if bs is not None:
        title = self.safeGet(bs, site.titleTag)
        body = self.safeGet(bs, site.bodyTag)
        if title != '' and body != '':
            content = Content(url, title, body)
            content.print()
```

- Parse(site,url)
- Def parse: url 받아 콘텐츠 추출하는 함수 :

```
crawler = Crawler()
```

```
siteData = [
    ['O'Reilly Media', 'http://oreilly.com', 'h1', 'section#product-description'],
    ['Reuters', 'http://reuters.com', 'h1', 'div.StandardArticleBody_body_1gnLA'],
    ['Brookings', 'http://www.brookings.edu', 'h1', 'div.post-body'],
    ['New York Times', 'http://nytimes.com', 'h1', 'div.StoryBodyCompanionColumn div p']
]
```

```
websites = []
for row in siteData:
    websites.append(Website(row[0], row[1], row[2], row[3]))
```

```
crawler.parse(websites[0], 'http://shop.oreilly.com/product/0636920028154.do')
crawler.parse(websites[1], 'http://www.reuters.com/article/us-usa-epa-pruitt-idUSKBN19W2')
crawler.parse(websites[2], 'https://www.brookings.edu/blog/techtank/2016/03/01/idea-to-re')
crawler.parse(websites[3], 'https://www.nytimes.com/2018/01/28/business/energy-environme')
```

- 사이트에 대한 데이터 siteData
- siteData 정보를 -> website 에 저장해서 for문으로 각 정보를 저장
- Row[0] 웹사이트 이름, row[1]: url, row[2]: 태그, row[3] : Css

```
URL: https://www.brookings.edu/blog/techtank/2016/03/01/idea-to-re
TITLE: Idea to Retire: Old methods of policy education
Idea to Retire: Old methods of policy education
BODY:
```

Public policy and public affairs schools aim to train competent principles that gird our economic and political systems to policy schools provide professional training. They are quite im to train the next generation of academics. As professions

```
class Content:
    """Common base class for all articles/pages"""

    def __init__(self, topic, url, title, body):
        self.topic = topic
        self.title = title
        self.body = body
        self.url = url

    def print(self):
        """
        Flexible printing function controls output
        """
        print('New article found for topic: {}'.format(self.topic))
        print('URL: {}'.format(self.url))
        print('TITLE: {}'.format(self.title))
        print('BODY:\n{}'.format(self.body))

class Website:
    """Contains information about website structure"""

    def __init__(self, name, url, searchUrl, resultListing, resultUrl, absoluteUrl, titleTag, bodyTag):
        self.name = name
        self.url = url
        self.searchUrl = searchUrl
        self.resultListing = resultListing
        self.resultUrl = resultUrl
        self.absoluteUrl = absoluteUrl
        self.titleTag = titleTag
        self.bodyTag = bodyTag
```

```
New article found for topic: python
URL: Inside the Pentagon's Secret Afghan Spy Machine
TITLE:
The Pentagon's top researchers have rushed a classified and controversial report, previously undisclosed as a war-zone surveillance effort to glean clues about Afghan instability.
```

```
BODY:
https://www.brookings.edu/opinions/inside-the-pentagons-secret-afghan-spy-machine/
```

```
class Crawler:

    def getPage(self, url):
        try:
            req = requests.get(url)
        except requests.exceptions.RequestException:
            return None
        return BeautifulSoup(req.text, 'html.parser')

    def safeGet(self, pageObj, selector):
        childObj = pageObj.select(selector)
        if childObj is not None and len(childObj) > 0:
            return childObj[0].get_text()
        return ''

    def search(self, topic, site):
        """
        Searches a given website for a given topic and records all pages found
        """
        bs = self.getPage(site.searchUrl + topic)
        searchResults = bs.select(site.resultListing)
        for result in searchResults:
            url = result.select(site.resultUrl)[0].attrs['href']
            # Check to see whether it's a relative or an absolute URL
            if site.absoluteUrl:
                bs = self.getPage(url)
            else:
                bs = self.getPage(site.url + url)
            if bs is None:
                print('Something was wrong with that page or URL. Skipping!')
                return
            title = self.safeGet(bs, site.titleTag)
            body = self.safeGet(bs, site.bodyTag)
            if title != '' and body != '':
                content = Content(topic, title, body, url)
                content.print()
```

- 주어진 검색어로 주어진 웹사이트 검색해 결과 페이지 모두 기록

```
crawler = Crawler()

siteData = [
    ['O'Reilly Media', 'http://oreilly.com', 'https://ssearch.oreilly.com/article/product-result', 'p.title a', True, 'h1', 'section#product-listing'],
    ['Reuters', 'http://reuters.com', 'http://www.reuters.com/search/news', 'h3.search-result-title a', False, 'h1', 'div.StandardArticleListElement'],
    ['Brookings', 'http://www.brookings.edu', 'https://www.brookings.edu/databases/policybriefs/list-content.asp?cat=1', 'h4.title a', True, 'h1', 'div.position']
]

sites = []
for row in siteData:
    sites.append(Website(row[0], row[1], row[2], row[3], row[4], row[5], row[6], row[7]))

topics = ['python', 'data science']
for topic in topics:
    print('GETTING INFO ABOUT: ' + topic)
    for targetSite in sites:
        crawler.search(topic, targetSite)
```

- website 데이터, 검색어 목록
- 웹사이트 전체에 대한 검색어 전체를 검색하는 이중 루프 추가
- 특정 웹사이트 및 주제에 대한 검색 페이지로 이동
- 해당 페이지에 나열된 결과 URL 모두 추출하는 search 함수

- Topics 리스트 항목 반복, 각 스크립트 하기 전 어떤 주제에 대한 스크랩인지 알림
- Sites 리스트 사이트 반복, 외부 루프에서 지정한 검색어로 각 사이트 스크랩

### 4.3 크롤러 구성

- 4.3.1 검색을 통한 사이트 크롤링
- 웹사이트를 크롤링하는 가장 쉬운 방법 : 검색창 이용 (키워드, 주제 검색, 검색 결과 목록 수집)
- 사이트 검색 결과
  - 사이트 URL 에 검색어 삽입-> 검색 결과
  - `http://example.com?search=MyTopic`
  - URL 첫번째 부분 Website 객체 속성으로 저장 ex.  
`http://example.com?search=`
  - 그 뒤 검색어 연결은 간단 `MyTopic`
- 검색 결과 페이지
  - 링크 목록 형태로 제공되는 형태가 대부분
  - `<span class='result'>` 같은 태그에 둘러싸여 있음
  - 이런 태그 형식도 Website 객체 속성으로 저장 가능
- 결과 링크
  - `/articles/page.html` 같은 상대 URL <또는>
  - `https://example.com/articles/page.html` 같은 절대 URL
  - 절대 URL이 필요한지 상대 URL 이 필요한지 역시 Website 객체의 속성으로 저장 가능

### Crawling through sites with search

```
class Content:
    """Common base class for all articles/pages"""

    def __init__(self, topic, url, title, body):
        self.topic = topic
        self.title = title
        self.body = body
        self.url = url

    def print(self):
        """
        Flexible printing function controls output
        """
        print('New article found for topic: {}'.format(self.topic))
        print('URL: {}'.format(self.url))
        print('TITLE: {}'.format(self.title))
        print('BODY:\n{}'.format(self.body))
```

- Class Content:
- Url 속성 추가, 콘텐츠의 위치 추적

```
class Website:
    """Contains information about website structure"""

    def __init__(self, name, url, searchUrl, resultListing, resultUrl, absoluteUrl, titleTag, bodyTag):
        self.name = name
        self.url = url
        self.searchUrl = searchUrl
        self.resultListing = resultListing
        self.resultUrl = resultUrl
        self.absoluteUrl = absoluteUrl
        self.titleTag = titleTag
        self.bodyTag = bodyTag
```

- searchURL: URL에 검색어 추가하는 경우 검색 결과 어디에서 얻는지 정의
- resultListing : 각 결과에 대한 정보 담은 박스
- resultUrl: 결과에서 정확한 URL 추출할 때 사용할 태그 정보
- absoluteUrl: 검색 결과가 절대 URL인지 상대 URL 인지 알려주는 Boolean 값

# 4장 Web crawling model

## 4.3.2 링크를 통한 사이트 크롤링

- 특정 url 패턴과 일치하는 리크를 모두 따라갈 수 있는 유연한 크롤러
- 특정 검색 결과나 페이지 목록에 국한되지 않고 사이트 전체에서 데이터를 수집해야 하는 프로젝트에 활용 가능
- 사이트의 페이지가 적절히 구조화되지 않았거나 광범위하게 분산된 경우에도 효과적
- 이런 타입의 크롤러 검색 페이지 크롤링하는 크롤러와 달리, 링크 위치를 확인하는 구조화된 방법이 필요하지 않아, website 객체에 검색 페이지에 관한 속성을 둘 필요가 없음
- 찾아야 할 링크의 위치가 없어 어떤 종류의 페이지 선택할지 지정하는 규칙 필요
- 대상 URL 에 대한 정규 표현식인 TargetPatter 과 Boolean 변수 absoluteUrl 사용

## Crawling Sites through Links ¶

```
class Website:
    def __init__(self, name, url, targetPattern, absoluteUrl, titleTag, bodyTag):
        self.name = name
        self.url = url
        self.targetPattern = targetPattern
        self.absoluteUrl = absoluteUrl
        self.titleTag = titleTag
        self.bodyTag = bodyTag

class Content:
    def __init__(self, url, title, body):
        self.url = url
        self.title = title
        self.body = body

    def print(self):
        print('URL: {}'.format(self.url))
        print('TITLE: {}'.format(self.title))
        print('BODY: \n{}'.format(self.body))
```

```
class Crawler:
    def __init__(self, site):
        self.site = site
        self.visited = []

    def getPage(self, url):
        try:
            req = requests.get(url)
        except requests.exceptions.RequestException:
            return None
        return BeautifulSoup(req.text, 'html.parser')

    def safeGet(self, pageObj, selector):
        selectedElems = pageObj.select(selector)
        if selectedElems is not None and len(selectedElems) > 0:
            return '\n'.join([elem.get_text() for elem in selectedElems])
        return ''

    def parse(self, url):
        bs = self.getPage(url)
        if bs is not None:
            title = self.safeGet(bs, self.site.titleTag)
            body = self.safeGet(bs, self.site.bodyTag)
            if title != '' and body != '':
                content = Content(url, title, body)
                content.print()

    def crawl(self):
        """
        Get pages from website home page
        """
        bs = self.getPage(self.site.url)
        targetPages = bs.findAll('a', href=re.compile(self.site.targetPattern))
        for targetPage in targetPages:
            targetPage = targetPage.attrs['href']
            if targetPage not in self.visited:
                self.visited.append(targetPage)
                if not self.site.absoluteUrl:
                    targetPage = '{}{}'.format(self.site.url, targetPage)
                self.parse(targetPage)
```

- `reuters = Website('Reuters', 'https://www.reuters.com', '^(/article/)',`
- `False, 'h1', 'div.StandardArticleBody_body_1gnLA')`
- `crawler = Crawler(reuters)`
- `crawler.crawl()`

- **Class Crawler:**
- 각 사이트 홈 페이지에서 시작하여 내부 링크를 찾고
- 발견된 각 내부 링크의 내용을 구문 분석
- **Def crawl:** 사이트 홈페이지에서 페이지를 가져옴

```
GETTING https://www.reuters.com
GETTING https://www.reuters.com/article/us-usa-trump-5g/trump-national
1FH103
URL: https://www.reuters.com/article/us-usa-trump-5g/trump-national
103
TITLE: Trump security team sees building U.S. 5G network as option
BODY:
WASHINGTON (Reuters) - President Donald Trump's national security
ying on U.S. phone calls that include the government building a sup
I said on Sunday. The official, confirming the gist of a report fro
I in the administration and was six to eight months away from being
is aimed at addressing what officials see as China's threat to U.S.
tion has taken a harder line on policies initiated by predecessor B
```

# 4장 Web crawling model

## 4.3.3 여러 페이지 유형 크롤링

내부 링크 크롤링에 -> 뭘 얻게 될지 모를 수 있음

페이지 유형을 식별하는 방법 있음

- URL에 따라
  - 블로그 게시물 URL 패턴, `https://example.com/blog/title-of-post`
- 특정 필드 존재 여부
  - 페이지에 날짜 있지만 작성자 이름 없다면 보도 자료로 분류
  - 제목, 주 이미지, 가격 있지만 주요 콘텐츠 없다면 제품 페이지
- 페이지를 식별할 수 있는 특정 태그 여부
  - `<div id="relatedproducts">` 요소, 제품 페이지라고 판단
  - 페이지 유형 추적 -> 여러 유형의 페이지 객체를 파이썬으로 만들
- 페이지가 모두 비슷하다면 기존 Website객체에 `pageType` 속성 추가함
- 스크랩하는 페이지/콘텐츠 서로 다르면, 각 페이지 유형에 대해 새 객체 생성

```
class Product(Website):
    """Contains information for scraping a product page"""

    def __init__(self, name, url, titleTag, productNumber, price):
        Website.__init__(self, name, url, titleTag)
        self.productNumberTag = productNumberTag
        self.priceTag = priceTag

class Article(Website):
    """Contains information for scraping an article page"""

    def __init__(self, name, url, titleTag, bodyTag, dateTag):
        Website.__init__(self, name, url, titleTag)
        self.bodyTag = bodyTag
        self.dateTag = dateTag
```

## Crawling multiple page types

```
class Website:
    """Common base class for all articles/pages"""

    def __init__(self, name, url, titleTag, bodyTag):
        self.name = name
        self.url = url
        self.titleTag = titleTag
        self.bodyTag = bodyTag
        self.pageType = pageType
```

- Class Product:
- 제품 페이지 스크랩에 필요한 정보를 저장하는 클래스
- Ex. `productNumber`, `price` 속성 추가
- Class Article:
- 기사 페이지 스크랩에 필요한 정보를 저장하는 클래스
- Ex. `Body`, `date` 속성

- 4.4 웹 크롤러 모델
- 여러 도메인, 여러 소스에서 유사한 데이터 수집할 때 **일반화** 시도해야 함!