

## 09 웹 다루기

# 09.01

## 웹 이해하기

웹은 언제나 접근가능한 공간입니다. 친구들과의 대화, 재미있는 이야기와 사진이 있고 또 내가 알고 싶어하는 내용이 가득한 재미있는 공간입니다. 이번 시간에는 프로세싱을 사용해 더욱 재미있고 편하게 웹을 사용하는 체험을 해봅시다. 언제나 그렇듯 아주 조금의 기술을 배워야 할 필요가 있고 그것보다 더 중요한 것은 여러분의 상상력입니다.

### 클라이언트와 서버 이해하기

---

우선 클라이언트와 서버라는 용어를 이해해볼까요?

서버라고 하면 큰 컴퓨터가 생각나시나요? 뭐 그런 컴퓨터를 서버로 쓰는 것이 자연스럽기는 하지만 여러분이 사용하는 컴퓨터도 서버로 동작할 수 있으니 크기가 중요한 것은 아닙니다.

중요한 것은 서버가 하는 역할입니다. 클라이언트는 다른 프로그램 혹은 컴퓨터에 어떤 서비스를 요청하는 컴퓨터나 프로그램을 지칭합니다. 서버는 그 요청에 대해 응답을 해주는 프로그램이나 컴퓨터를 말합니다.

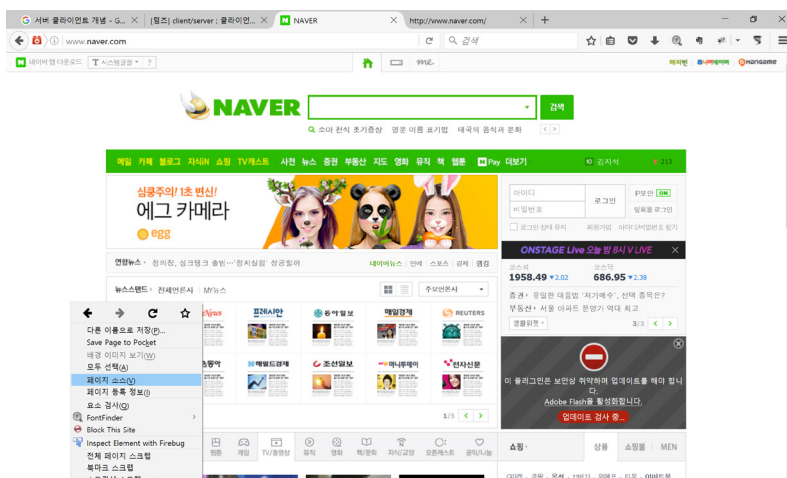
웹 브라우저를 띄워 네이버 주소를 입력하면 여러분의 컴퓨터는 네이버 서버에 웹페이지를 요청하게 됩니다. 응답에 반응한 네이버 서버가 여러분의 컴퓨터에 html 코드를 보내주고 웹 브라우저는 그 코드를 해석해 화면에 글자와 사진을 배치합니다.

이제 개념이 간단하게 정리되었나요? 내가 요청하면 클라이언트, 내가 제공하면 서버. 우리가 웹을 이용할 때는 대부분 클라이언트로 웹서버에 특정 페이지를 요청하는 것입니다. 그러면 웹서버가 우리의 요청에 맞는 페이지를 내 컴퓨터에 전송해주고 웹브라우저는 전송받은 html 코드를 사람이 이해하기 쉬운 형태로 렌더링해서 보여줍니다.

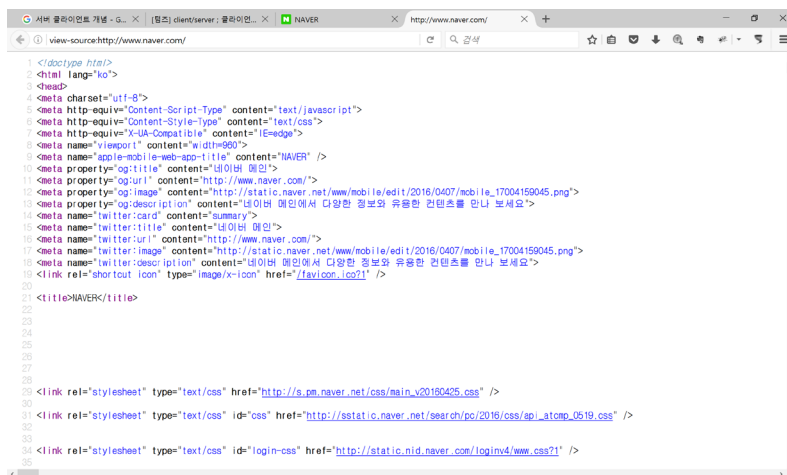
## html 이해하기

html 코드가 무엇인지 궁금하지 않으신가요? 어떻게 생긴 것인지 한번 보고 싶지 않으세요?

모두들 브라우저를 열어서 네이버에 접속해보세요. 그리고 소스보기를 해봅시다.



사람이 읽기는 불가능할 것 같은 복잡한 텍스트가 한 가득 보일 것입니다.



그렇다면 이 문서를 읽는데 도움을 청해야 하겠습니다. html 문서 분석을 위한 라이브러리 Jsoup을 소개합니다.

## 09.02

# Jsoup 라이브러리 이해하기

프로세싱은 엄밀하게 이야기하면 자바라는 프로그래밍 언어 위에 구현된 프로그래밍 언어입니다. 자바를 다루는 일이 생각보다 어렵고 학습부담이 많기 때문에 처음 프로그래밍을 접하는 분들을 위해 많은 부분을 간략화하고 단순하게 유지한 언어라고 이해하시면 좋을 것 같습니다.

하지만 여전히 자바의 영향권 아래에 있는데 사실 이것은 꽤 좋은 일이기도 합니다. 왜냐하면 프로세싱이 할 수 없는 일을 자바는 할 수 있는 경우가 많기 때문입니다.

혹시 프로세싱으로 하기 어려운 일이 있다면 자바 라이브러리로 이미 구현되었는지 확인하는 습관을 기르는 것도 좋습니다. 자바의 넓은 사용자 층과 오랜 역사는 정말 없는게 없다는 정도로 방대한 라이브러리를 자랑합니다.

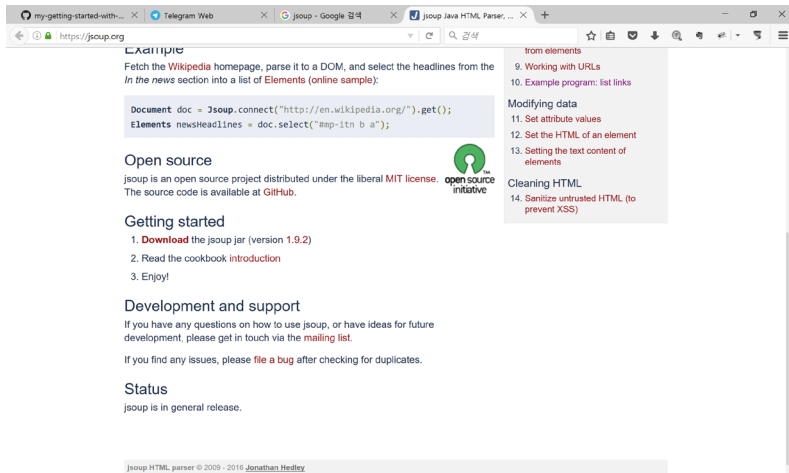
웹페이지를 분석하는 것도 마찬가지입니다. 사용하기 쉬운 자바 라이브러리 Jsoup이 이미 있으니 우리는 고민하지 말고 Jsoup을 사용하도록 합시다.

Jsoup은 Java로 작성된 HTML 파서입니다. 아까 네이버 소스코드 보기로 확인했던 html코드를 분석하고 구조화하는 라이브러리로 자료를 추출하거나 가공하는데 편리하게 사용할 수 있는 라이브러리입니다.

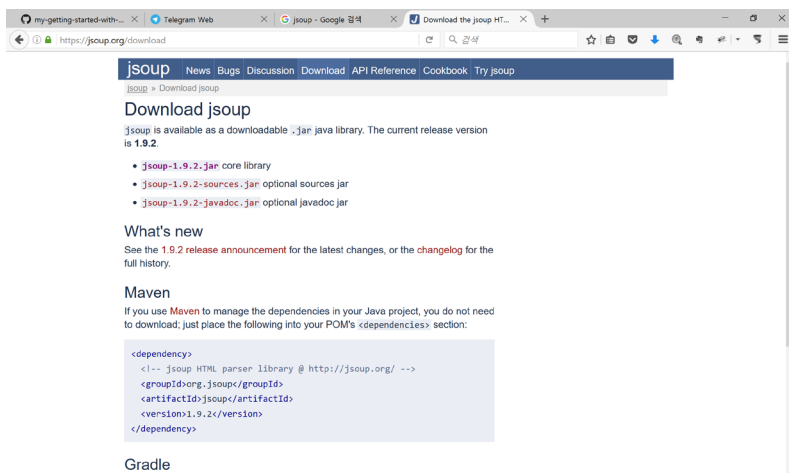
그런데 왜 Jsoup이나? 아마도 파이썬의 유명한 웹사이트 분석 라이브러리 BeautifulSoup에 영향을 받은 것으로 보입니다. 유명한 라이브러리로 저도 파이썬을 사용해 html코드를 분석할 때 사용합니다. 비슷한 예로 구글에서 만든 Google Gumbo가 있습니다. 라이브러리 이름에 미국 남부 음식 Gumbo를 붙였습니다. 프로그래머들이 생각보다 유머있는 사람들입니다. 이런 것을 보면 말입니다.

## 설치

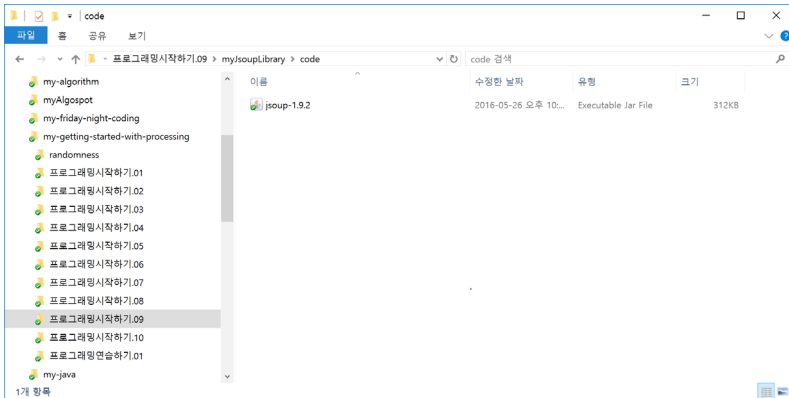
Jsoup을 설치하려면 아래의 사이트를 방문하세요.



Download 항목을 클릭한 다음 아래의 페이지에서 jsoup-[current-version].jar을 다운받으세요.



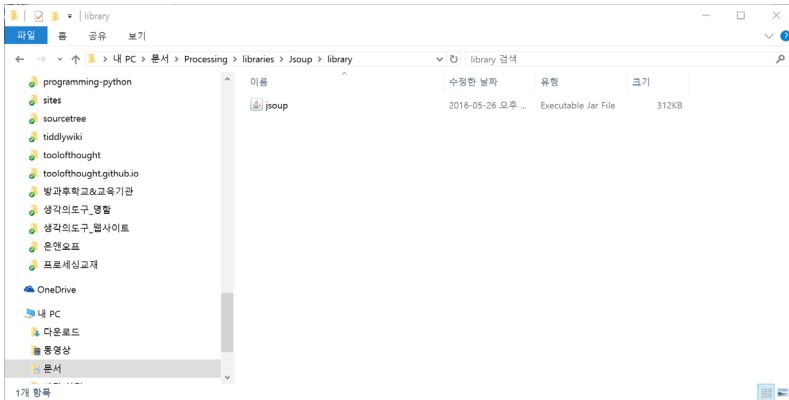
이제 jsoup-[current-version].jar 파일을 이미지를 프로세싱 스케치에 추가할 때와 마찬가지로 드래그해서 편집창에 놓으세요. 탐색기를 실행시키면 code 폴더 아래 복사된 jar 파일을 확인할 수 있습니다.



설치가 끝났습니다. 생각보다 별 것 아닙니다. 자바 외부라이브러리의 설치는 모두 이렇게 실행됩니다.

매번 라이브러리를 사용하는 스케치에 jar 파일을 추가하는 대신 프로세싱 폴더에 추가하는 방법도 있습니다.

따로 정하지 않았다면 프로세싱 라이브러리는 사용자의 My Document 폴더 안에 위치합니다.



## Hello, Jsoup

Jsoup 라이브러리를 사용하려면 코드의 시작부분에 몇 줄을 추가해야 합니다.

```
import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Element;
import org.jsoup.select.Elements;
```

모든 것을 이해하려고 하실 필요는 당연히 없습니다. 위의 네 줄은 기계적으로 입력한다고 이해하는 것으로 충분한 내용이니 복사해서 사용하는

것으로 충분합니다. 모든 것을 다 알 수는 없다는 것, 내가 이해하지 못하는 것을 이용하는데 양심의 가책을 느끼는 것은 바보스러운 일입니다. 단언컨데 세상의 어떤 프로그래머도 모든 것을 알고 라이브러리를 사용하지는 않습니다. 다만 설명서를 읽고 예제를 보고 라이브러리를 사용하는 것으로 충분합니다. 한 번에 모든 것을 이해하려 하지 마세요.

Jsoup, Document, Element, Elements는 대문자로 시작하고 있는데 관용적으로 클래스라고 이해할 수 있습니다. 아직 OOP에 대해 정식으로 배우지는 않았지만 이미지를 다룰 때 PImage라는 타입을 본 적이 있습니다. Jsoup, Document, Element, Elements 등을 다룰 때 PImage의 기억을 떠올려 보세요. get() 메서드를 어떻게 호출했는지를 떠올려보시고 이런 문법이 다른 클래스에 동일하게 적용된다는 점을 이해하시면 많은 도움이 될 것입니다. 앞으로 프로세싱을 대하면서 대문자로 시작되는 코드를 보게 되면 클래스가 아닌지 의심해보세요. 높은 확률로 그럴 것입니다.

이제 Jsoup을 이용한 첫번째 작업을 수행해 봅시다.

## 웹페이지 데이터 분석하기

걸그룹의 사진이 필요합니다. 어떤 펜 사이트에 사진이 잔뜩 있지만 너무 많아서 손으로 하나씩 다운받기는 불가능 합니다. 어떻게 해야할까요?

여러분 이런 문제야 말로 정말로 생활 속의 프로그래밍이라고 부를만 하지 않을까요?

문제 해결을 위해 전략을 세워봅시다.

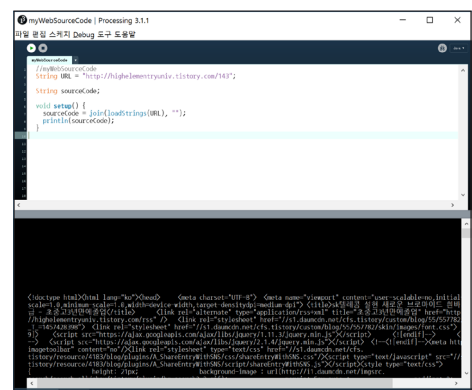
0. 웹사이트 소스코드를 불러옵니다.
1. 소스코드를 분석해 이미지 파일 주소만을 따옵니다. - Jsoup
2. 이미지 파일을 저장합니다. - loadImage(), save() 메서드
3. 즐겁게 감상합니다.

한 단계씩 진행해볼까요?

## 웹사이트 소스코드 불러오기

```
//myWebSourceCode
String URL = "http://highementryuniv.tistory.com/143";

String sourceCode;
```



```
void setup() {
    sourceCode = join(loadStrings(URL), "");
    println(sourceCode);
}
```

## 사진링크 따오기

이제 정말 Jsoup을 사용해 html문을 분석합니다. 소스를 살펴보니 이미지 파일은 모두 `img` 태그 안에 위치합니다. 이미지의 위치는 `img` 태그의 `src` 속성에 지정되어 있습니다. 그렇다면 답은 간단합니다. `img` 태그의 `src` 속성값을 모두 가지고 옵시다.

코드가 약간 복잡하게 되어 있는데 차근차근 종이에 써가면서 구조를 분석해봅시다.

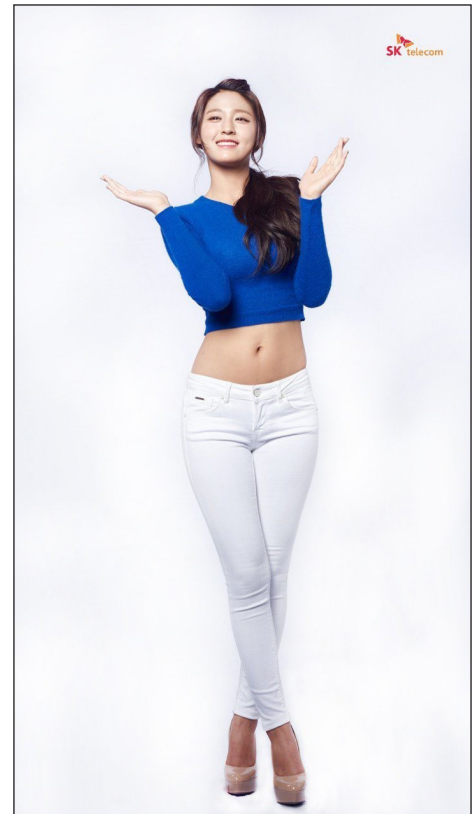
```
//myWebSourceCodeImgSrc
import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Element;
import org.jsoup.select.Elements;

String URL = "http://highentryuniv.tistory.com/143";
String sourceCode;
ArrayList<String> links = new ArrayList<String>();
ArrayList<PImage> images = new ArrayList<PImage>();

void setup() {
    size(720, 1280);
    frameRate(1);
    sourceCode = join(loadStrings(URL), "");
    Document doc = Jsoup.parse(sourceCode);
    Elements imgLinks = doc.select("div.tt_article_
useless_p_margin img");

    for (Element image : imgLinks) {
        links.add(image.attr("src"));
        images.add(loadImage(image.attr("src"),
"jpg"));
    }
}

void draw() {
    image(images.get(frameCount % images.size()),
0, 0);
}
```



myWebSourcecodeImgSrc



## 네이버 실시간 급상승 검색어 따오기

이제 네이버 첫 화면의 실시간 급상승 검색어를 가져오겠습니다. 복습하는 차원에서 작업의 큰 그림을 그려보면 html 소스 → Document → Elements → Element 순으로 작업이 진행됩니다. 복잡해보이지만 단계별로 생각해보면 합리적인 수준이지요.

네이버 첫 화면 소스보기를 해봅시다.

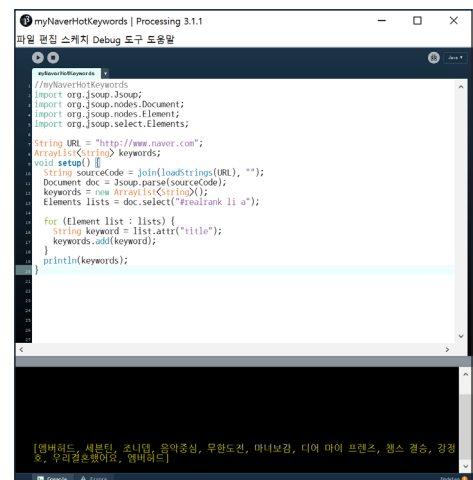
```
<ol style="margin-top:0px;" id="realrank">
  <li>...</li>
  <li>...</li>
  <li>...</li>
</ol>
```

실시간 급상승 검색어는 realrank라는 id로 구별이 가능하고 `li` 태그 안에 정보가 위치합니다. `id`까지 지정해두었다면 분석은 식은 죽 먹기입니다. `id`는 그 페이지 안에서 유일하게 결정되는 속성이니까요. 키워드는 `id`가 `realrank`인 `ol` 태그 아래 `li` 태그 아래 `a` 태그에 저장되어 있습니다. `a` 태그의 `title` 속성을 따오면 쉽게 키워드명을 얻을 수 있겠습니다.

```
//myNaverHotKeywords
import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Element;
import org.jsoup.select.Elements;

String URL = "http://www.naver.com";
ArrayList<String> keywords;
void setup() {
  String sourceCode = join(loadStrings(URL), "");
  Document doc = Jsoup.parse(sourceCode);
  keywords = new ArrayList<String>();
  Elements lists = doc.select("#realrank li a");

  for (Element list : lists) {
    String keyword = list.attr("title");
    keywords.add(keyword);
  }
  println(keywords);
}
```



myNaverHotKeywords

이렇게 쉬워도 되는 겁니까? 네. 가끔은 이렇게 쉬울 때도 있는 겁니다.

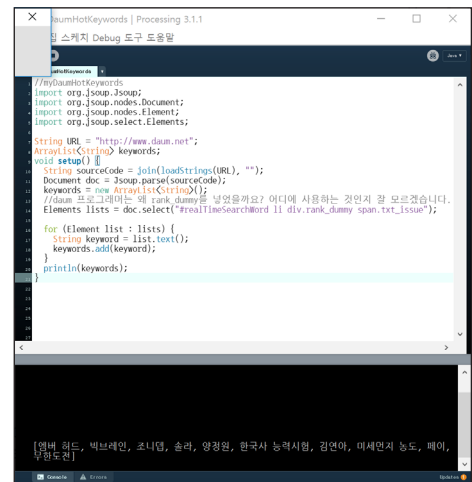
## 다음 실시간 급상승 검색어 따오기

다음이라고 다를 것이 있겠습니까? html구조만 잘 분석하면 네이버와 다를 것은 없습니다.

```
//myDaumHotKeywords
import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Element;
import org.jsoup.select.Elements;

String URL = "http://www.daum.net";
ArrayList<String> keywords;
void setup() {
    String sourceCode = join(loadStrings(URL), "");
    Document doc = Jsoup.parse(sourceCode);
    keywords = new ArrayList<String>();
    //daum 프로그래머는 왜 rank_dummy를 넣었을까요? 어디에
    //사용하는 것인지 잘 모르겠습니다. 이렇게 구분하지 않으면 키워드
    //가 두 번씩 반복됩니다.
    Elements lists = doc.
    select("#realTimeSearchWord li div.rank_dummy
    span.txt_issue");

    for (Element list : lists) {
        String keyword = list.text();
        keywords.add(keyword);
    }
    println(keywords);
}
```



myDaumHotKeywords

## 09.03

# Selenium 라이브러리 이해하기

이제 두번째 자바 라이브러리 Selenium에 대해 알아볼 시간입니다. Jsoup이 html파서라고 한다면 Selenium은 웹브라우저를 자동화하는 라이브러리입니다. 여러분이 웹브라우저를 대하며 하는 동작을 자동화 할 수 있습니다. 무슨 소리냐구요? 여러분이 버튼을 클릭하거나 키보드로 암호를 입력하는 것과 같은 행동을 프로그래밍으로 제어할 수 있습니다.

원래는 웹페이지를 테스트하는데 사용되는 라이브러리지만 사용하다보니 웹브라우저 automation 라이브러리로 사용하는 것이 편해서 저는 이 라이브러리를 자주 사용하고 있습니다.

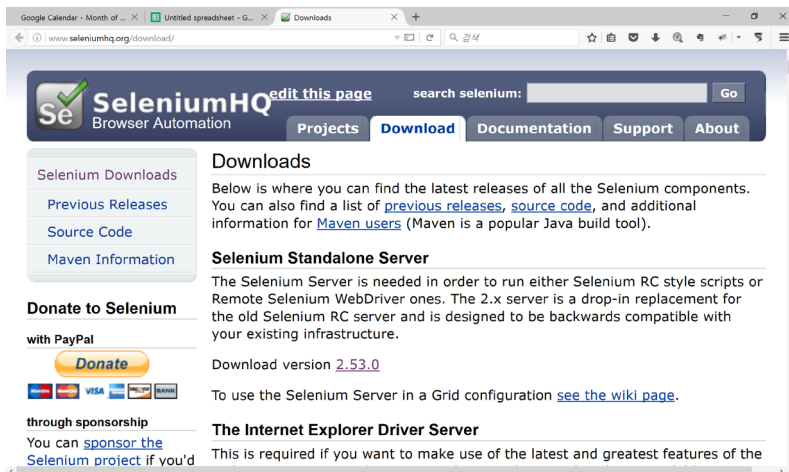
예를 들어 날짜를 입력하면 그 날짜에 해당하는 데이터를 보여주는 웹사이트 있다고 합시다. 하루 이틀 정도는 손으로 처리할 수 있지만 한 달 혹은 일년 데이터를 뽑아야 한다면 제정신이 아니고서야 실수없이 손으로 자료를 처리하기란 불가능에 가깝습니다. 이럴 때 selenium을 사용해서 날짜입력을 자동으로 하고 Jsoup으로 자료를 처리하게 하면 실수없이 자료를 수집할 수 있습니다.

어떤가요 아직도 감이 오지 않습니까? 괜찮습니다. 예제를 따라가면 무슨 소리인지 알 수 있습니다. 두려워마시고 일단 프로그램을 실행해봅시다. 조금씩 기능을 추가하다 보면 우리가 원하는 프로그램을 만들 수 있습니다.

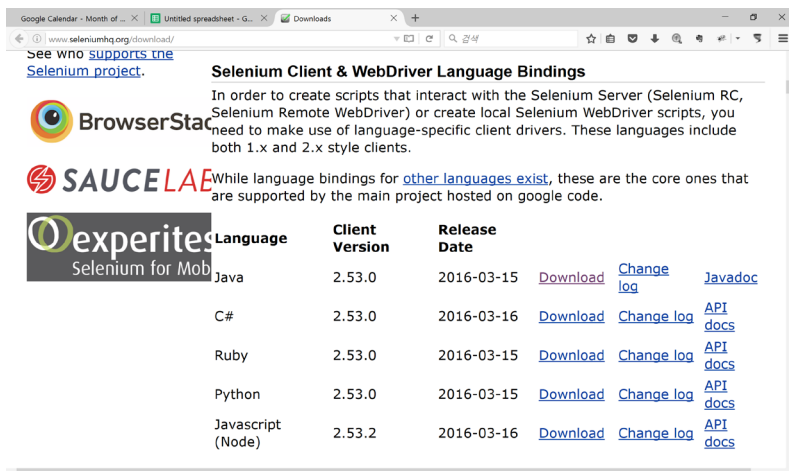
## 설치

두 가지 파일을 다운받아야 합니다.

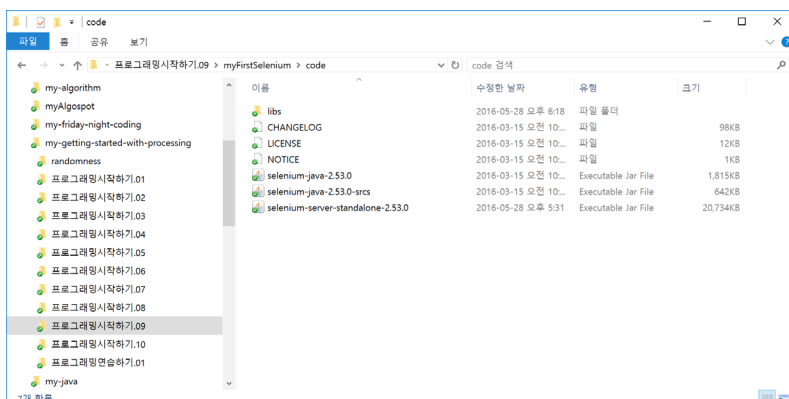
첫번째는 Selenium Standalone Server파일입니다.



두번째는 자바버전 Selenium Client & WebDriver Language Binding입니다. 자바 버전을 선택하세요.



다운받은 파일의 압축을 풀고 모든 파일을 스케치 창 위로 드래그 하면 스케치 폴더의 code폴더에 모두 저장된 것을 확인할 수 있습니다. 아니면 code폴더를 만들고 모두 저장하는 것도 방법입니다. 편한 방법을 따르세요.



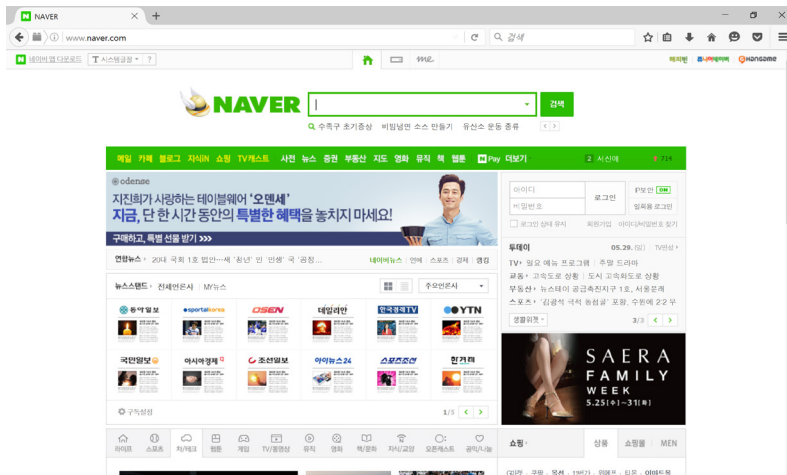
## 첫번째 seleniumproject: 창 띄우기

사실 selenium이 어떤 것인가 아무런 감이 오지 않습니다. 그럴 때면 일단 실행해보는 것이 좋습니다.

아래의 코드를 한 번 살펴보니 driver라는 객체를 만들어 네이버 웹페이지를 열고 있습니다. 실행하면 어떤 일이 생길까요?

```
//myFirstSelenium
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.
FirefoxDriver;

void setup() {
    WebDriver driver = new FirefoxDriver();
    driver.get("http://www.naver.com");
}
```



보셨습니까? 파이어폭스 창이 새로 생기고 네이버에 접속합니다. 아니 이것이 무엇입니까? 파이어폭스 창을 새로 띄우다니요? 프로세싱이 뭐 이런 것까지 할 수 있는 것입니까?

네. 프로세싱 혹은 자바는 대단합니다. 이제부터라도 열심히 응원하고 사용합시다.

영화진흥위원회의 박스오피스 데이터에 늘 관심이 많았습니다. 위원회가 API를 공개하기도 했지만 제한이 있기 때문에 모든 데이터가 공개되는 사이트를 분석해서 원자료를 확보합니다.

[illegible]

현재 주소를 바로 받아오면 좋겠지만 몇 가지 해결해야 할 사항이 있습니다. 우선 관심있는 날짜를 입력해야 하는 것이 첫번째입니다. selenium을 활용해 원하는 날짜를 입력하는 작업을 추가할 것입니다.

두번째 문제는 모든 영화가 리스트에 나와있지 않습니다. 리스트의 끝에 있는 더보기 버튼을 클릭해야만 추가로 10개의 영화가 리스트에 추가됩니다. 이런 상태라면 Jsoup으로 바로 html 원문을 읽어도 소용이 없게 됩니다. selenium을 활용해서 모든 영화가 리스트에 추가될 때까지 "더보기" 버튼을 클릭하는 작업을 추가해야 합니다.

[illegible]

분석이 끝났으니 이제 selenium을 열심히 굴러볼 차례입니다. selenium과 Jsoup이 함께하면 두려울 것이 없습니다. 날짜는 selenium으로 자동입력하고 웹서버가 입력된 날짜의 페이지를 출력하면 이를 Jsoup을 이용해 필요한 정보만 잘라내겠습니다.

## getTable() 함수

마우스 조작과 키보드 입력을 selenium에 부탁해서 필요한 테이블을 얻읍시다.

```
//myGetMovieTable
import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Element;
import org.jsoup.select.Elements;

import org.openqa.selenium.By;
import org.openqa.selenium.Keys;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.
FirefoxDriver;

String URL = "http://www.kobis.or.kr/kobis/
business/stat/boxs/findDailyBoxOfficeList.do";
String DATE = "2016-05-26";

void setup() {
    println(getTable());
}

Elements getTable() {

    WebDriver driver = new FirefoxDriver();
    driver.get(URL);

    //date input
    WebElement inputDate = driver.findElement(By.
xpath("//input[@id='sSearchTo']"));
    for (int i = 0; i < 10; i++) inputDate.
sendKeys(Keys.BACK_SPACE);
    inputDate.sendKeys((DATE));

    //click search button
    WebElement searchButton = driver.
findElement(By.xpath("//a[@
onclick=\"chkform('search'); return false;\"]"));
    searchButton.click();

    //click button to expand list
    WebElement moreButton = driver.findElement(By.
id("btn_0"));
    while (moreButton.getAttribute("class").
equals("btn_on")) {
        moreButton.click();
    }

    //pass html source to Jsoup
```

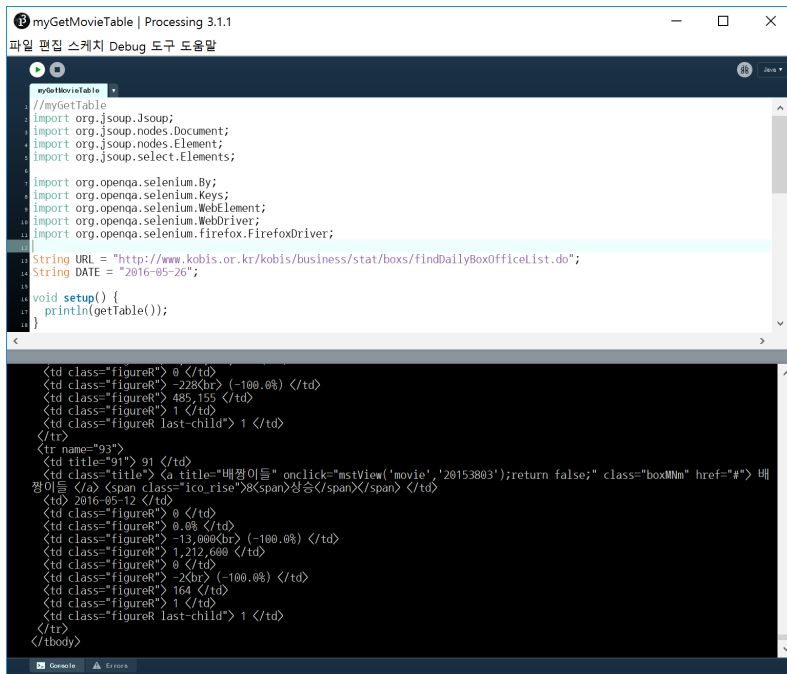
```

Document doc = Jsoup.parse(driver.
getPageSource());

Elements table = doc.select("#tbody_0");

return table;
}

```



## getMovieTitle()

이제 Jsoup의 세상입니다. html소스를 분석해 원하는 데이터를 소스에서 떼어웁니다.

먼저 영화의 이름을 테이블에서 떼어웁시다.

```

//myGetMovieTitle
import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Element;
import org.jsoup.select.Elements;

import org.openqa.selenium.By;
import org.openqa.selenium.Keys;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.
FirefoxDriver;

String URL = "http://www.kobis.or.kr/kobis/
business/stat/boxs/findDailyBoxOfficeList.do";

```



```

String DATE = "2016-05-26";

ArrayList<String> titles = new ArrayList<String>();
void setup() {
    getMovieTitle();
    printArray(titles);
}

void getMovieTitle() {
    Elements table = getTable();
    for (Element entry : table) {
        titles.add(entry.select("tr td.title").
text());
    }
}

Elements getTable() {

    WebDriver driver = new FirefoxDriver();
    driver.get(URL);

    //date input
    WebElement inputDate = driver.findElement(By.
xpath("//input[@id='sSearchTo']"));
    for (int i = 0; i < 10; i++) inputDate.
sendKeys(Keys.BACK_SPACE);
    inputDate.sendKeys((DATE));

    //click search button
    WebElement searchButton = driver.
findElement(By.xpath("//a[@
onclick=\"chkform('search'); return false;\\"]"));
    searchButton.click();

    //click button to expand list
    WebElement moreButton = driver.findElement(By.
id("btn_0"));
    while (moreButton.getAttribute("class").
equals("btn_on")) {
        moreButton.click();
    }

    //pass html source to Jsoup
    Document doc = Jsoup.parse(driver.
getPageSource());

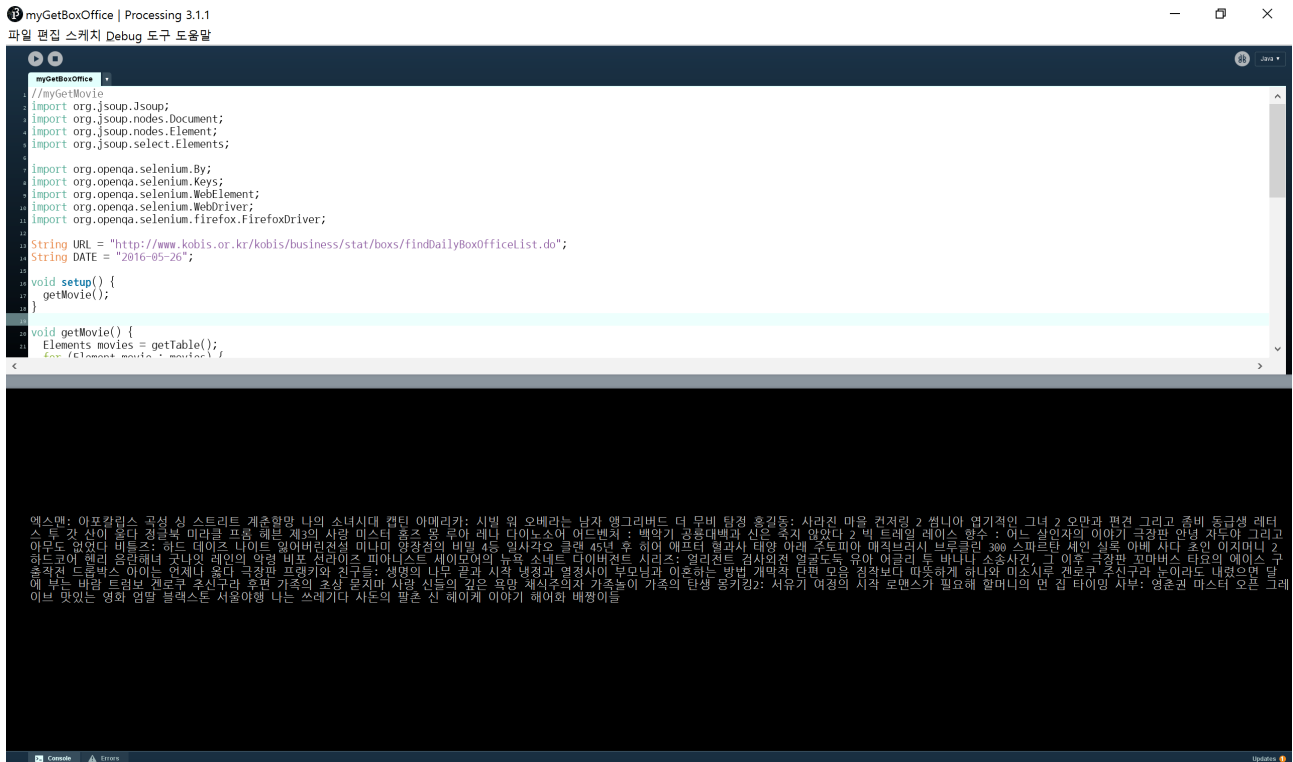
    Elements table = doc.select("#tbody_0");

    return table;
}

```

어떻습니까? 약간 손이 더 가는 작업을 했지만 본질적으로는 이전에 했던 과정과 다르지 않습니다. 복잡한 웹페이지에서 원하는 부분만 구하는 방

법을 배웠습니다.



## 흥행성적

이렇게 가면 서운하니 한가지만 추가해 봅시다. 영화 이름과 흥행성적을 함께 찾아봅시다. 이렇게 할 수도 있다는 정도로 만족하고 우선 구경하는 기분으로 아래의 예를 읽어봅시다.

getRevenue() 함수를 추가해 흥행성적을 가지고 옵니다.

계충화한 부분에 주목하세요. movies라는 묶음을 추가한 부분입니다. 이렇게 하면 데이터를 다루기가 쉬워집니다.

```
//myGetMovieTitleAndRevenue
import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Element;
import org.jsoup.select.Elements;

import org.openqa.selenium.By;
import org.openqa.selenium.Keys;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;

String URL = "http://www.kobis.or.kr/kobis/
business/stat/boxes/findDailyBoxOfficeList.do";
String DATE = "2016-05-26";
```

```

Elements table;

ArrayList<String> titles = new ArrayList<String>();
ArrayList<String> revenues = new
ArrayList<String>();

void setup() {
    table = getTable();
    getMovieTitle();
    getMovieRevenue();
    for (int i = 0; i < titles.size(); i++) {
        println(titles.get(i) + ": " + revenues.
get(i));
    }
}

void getMovieTitle() {
    Elements movies = table.select("tr");
    for (Element movie : movies) {
        titles.add(movie.select("td.title a").
text());
    }
}

void getMovieRevenue() {
    Elements movies = table.select("tr");
    for (Element movie : movies) {
        revenues.add(movie.select("td.figureR").
first().text());
    }
}

Elements getTable() {

    WebDriver driver = new FirefoxDriver();
    driver.get(URL);

    //date input
    WebElement inputDate = driver.findElement(By.
xpath("//input[@id='sSearchTo']"));
    for (int i = 0; i < 10; i++) inputDate.
sendKeys(Keys.BACK_SPACE);
    inputDate.sendKeys((DATE));

    //click search button
    WebElement searchButton = driver.
findElement(By.xpath("//a[@
onclick='chkform('search'); return false;\\"]"));
    searchButton.click();

    //click button to expand list
    WebElement moreButton = driver.findElement(By.

```

```

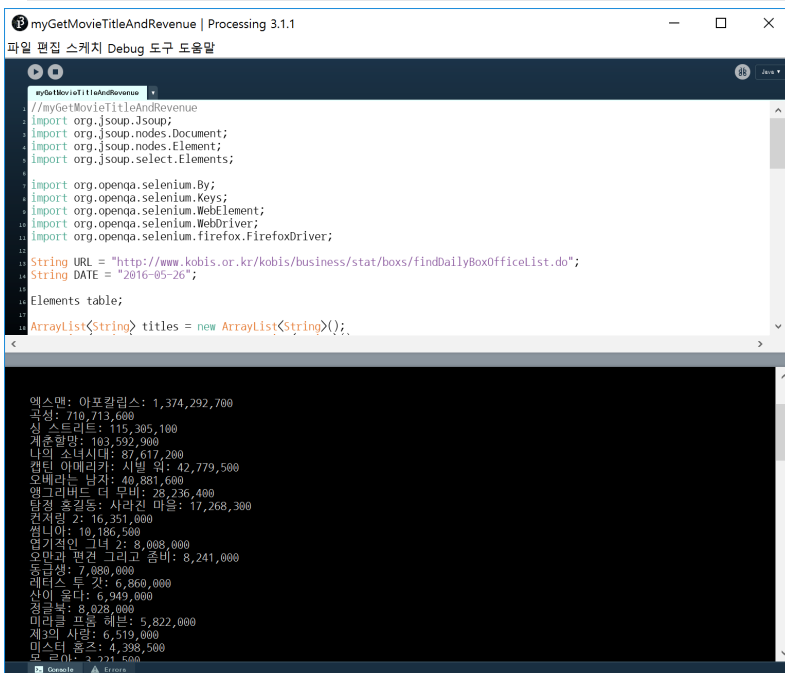
id("btn_0"));
    while (moreButton.getAttribute("class").
equals("btn_on")) {
        moreButton.click();
    }

    //pass html source to Jsoup
    Document doc = Jsoup.parse(driver.
getPageSource());

    Elements table = doc.select("#tbody_0");

    return table;
}

```



```

//myGetMovieTitleAndRevenue
import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Element;
import org.jsoup.select.Elements;

import org.openqa.selenium.By;
import org.openqa.selenium.Keys;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;

String URL = "http://www.kobis.or.kr/kobis/business/stat/boxes/findDailyBoxOfficeList.do";
String DATE = "2016-05-26";

Elements table;

ArrayList<String> titles = new ArrayList<String>();

```

콘솔 출력:

```

엑스맨: 아포칼립스: 1,374,292,700
국왕: 710,713,600
상: 스트리트: 115,205,100
계춘할망: 103,502,900
나의 소녀시대: 87,617,200
컴턴 아메리카: 시빌 워: 42,779,500
오백만은 님자: 40,881,600
앵그리버드 더 무비: 28,236,400
탐정 홍길동: 사라진 마을: 17,268,300
컨저링 2: 16,351,000
썬더: 10,186,500
원기결연: 귀녀 2: 8,000,000
오만과 편견: 그리고 좀비: 8,241,000
동급생: 7,000,000
레터스 투 갓: 6,860,000
산이 울다: 6,949,000
정글북: 8,028,000
마리콜 프롬 해설: 5,822,000
제3의 사랑: 6,519,000
미스터 홈즈: 4,398,500
모 로니: 3,721,500

```

어려울 것이 없습니다. 한 번에 하나씩 하는 한 번에 하나씩 하는 습관만 기르세요.

이런 기법을 웹크롤링(web crawling)이라고 부릅니다. 웹크롤링은 서버에 큰 부담을 주기 때문에 경우에 따라 웹크롤링을 실행하는 ip가 차단될 수도 있습니다. 물론 규모가 있는 웹서버의 경우 웹크롤링에 대한 대책이 마련되어 있어 큰 문제가 되지 않는 경우도 있지만 소규모 사이트의 경우 큰 부담으로 작용할 수도 있습니다. 따라서 원칙없이 무분별하게 웹크롤링을 실행하면 안 됩니다.

따로 원칙이 정해져 있는 것은 아니지만 우리 스스로에게 질문을 해보면 어느 정도까지 웹크롤링을 할 수 있는지 알 수 있습니다. 내가 운영하는 웹사이트에 어느 정도까지 웹크롤링을 허용할 것인지 스스로 답해보면 우리가 다른 사이트를 대상으로 어느 정도까지 공격적으로 웹크롤링을 할 수 있을 지 정하는 개인적인 기준을 정할 수 있습니다. 내가 싫어하는 일을 다른 사람에게 시키는 것을 옳지 않습니다.