# Python Internet

* div – indicates a division, or area, of the page.
* b – bolds any text inside.
* i – italicizes any text inside.
* table – creates a table.
* form – creates an input form.

## Request 函数包

**import** **requests ### download web page**

page **=** requests**.**get("http://dataquestio.github.io/web-scraping-pages/simple.html")

page

<Response [200]>：a status code starting with a 2 generally indicates success, and a code starting with a 4 or a 5 indicates an error.

## BeautifulSoup (Parsing a page) ###分解网页的内容

**from** **bs4** **import** BeautifulSoup

soup **=** BeautifulSoup(page**.**content, 'html.parser')

We can now **print out the HTML content of the page**, formatted nicely, using the prettify method on the BeautifulSoup object:

print(soup**.**prettify())

<!DOCTYPE html>

<html>

<head>

<title>

A simple example page

</title>

</head>

<body>

<p>

Here is some simple content for this page.

</p>

</body>

</html>

As all the tags are nested, we can move through the structure one level at a time. We can first select all the elements at the top level of the page using the children property of soup. Note that children returns a list generator, so we need to call the list function on it:

**list(soup.children)**

['html', '\n', <html>

<head>

<title>A simple example page</title>

</head>

<body>

<p>Here is some simple content for this page.</p>

</body>

</html>]

The above tells us that there are two tags at the top level of the page – the initial <!DOCTYPE html> tag, and the <html> tag. There is a newline character (\n) in the list as well. Let’s see what the type of each element in the list is:

[type(item) **for** item **in** list(soup**.**children)]

[bs4.element.Doctype, bs4.element.NavigableString, bs4.element.Tag]

The Tag object allows us to navigate through an HTML document, and extract other tags and text. You can learn more about the various BeautifulSoup objects [here](https://www.crummy.com/software/BeautifulSoup/bs4/doc/#kinds-of-objects).

We can now select the html tag and its children by taking the third item in the list:

html **=** list(soup**.**children)[2]

Each item in the list returned by the children property is also a BeautifulSoup object, so we can also call the children method on html.

Now, we can find the children inside the html tag:

list(html**.**children)

['\n', <head>

<title>A simple example page</title>

</head>, '\n', <body>

<p>Here is some simple content for this page.</p>

</body>, '\n']

As you can see above, there are two tags here, head, and body. We want to extract the text inside the p tag, so we’ll dive into the body:

body **=** list(html**.**children)[3]

Now, we can get the p tag by finding the children of the body tag:

list(body**.**children)

['\n', <p>Here is some simple content for this page.</p>, '\n']

We can now isolate the p tag:

p **=** list(body**.**children)[1]

Once we’ve isolated the tag, we can use the get\_text method to extract all of the text inside the tag:

p**.**get\_text()

'Here is some simple content for this page.'

## Finding all instances of a tag at once

What we did above was useful for figuring out how to navigate a page, but it took a lot of commands to do something fairly simple. If we want to extract a single tag, we can instead use the find\_all method, which will find all the instances of a tag on a page.

soup **=** BeautifulSoup(page**.**content, 'html.parser')

soup**.**find\_all('p')

[<p>Here is some simple content for this page.</p>]

Note that find\_all returns a list, so we’ll have to loop through, or use list indexing, it to extract text (get\_text):

soup**.**find\_all('p')[0]**.**get\_text()

'Here is some simple content for this page.'

If you instead only want to find the first instance of a tag, you can use the find method, which will return a single BeautifulSoup object:

soup**.**find('p')

<p>Here is some simple content for this page.</p>

### ‘find’的输出是beautifulsoup的object. ‘find\_all’的输出是一个数列。

## Searching for tags by class and id

**<html>**

**<head>**

**<title>**A simple example page**</title>**

**</head>**

**<body>**

**<div>**

**<p** class="inner-text first-item" id="first"**>**

First paragraph.

**</p>**

**<p** class="inner-text"**>**

Second paragraph.

**</p>**

**</div>**

**<p** class="outer-text first-item" id="second"**>**

**<b>**

First outer paragraph.

**</b>**

**</p>**

**<p** class="outer-text"**>**

**<b>**

Second outer paragraph.

**</b>**

**</p>**

**</body>**

**</html>**

We can access the above document at the URL http://dataquestio.github.io/web-scraping-pages/ids\_and\_classes.html. Let’s first download the page and create a BeautifulSoup object:

age **=** requests**.**get("http://dataquestio.github.io/web-scraping-pages/ids\_and\_classes.html")

soup **=** BeautifulSoup(page**.**content, 'html.parser')

soup

<html>

<head>

<title>A simple example page</title>

</head>

<body>

<div>

<p class="inner-text first-item" id="first">

First paragraph.

</p>

<p class="inner-text">

Second paragraph.

</p>

</div>

<p class="outer-text first-item" id="second">

<b>

First outer paragraph.

</b>

</p>

<p class="outer-text">

<b>

Second outer paragraph.

</b>

</p>

</body>

</html>

Now, we can use the find\_all method to search for items by class or by id. In the below example, we’ll search for any p tag that has the class outer-text:

soup**.**find\_all('p', class\_**=**'outer-text')

[<p class="outer-text first-item" id="second">

<b>

First outer paragraph.

</b>

</p>, <p class="outer-text">

<b>

Second outer paragraph.

</b>

</p>]

###比起’find\_all’，添加了一个限定条件---class。

In the below example, we’ll look for any tag that has the class outer-text:

soup**.**find\_all(class\_**=**"outer-text")

[<p class="outer-text first-item" id="second">

<b>

First outer paragraph.

</b>

</p>, <p class="outer-text">

<b>

Second outer paragraph.

</b>

</p>]

###两种find\_all的写法相同

We can also search for elements by id:

soup**.**find\_all(id**=**"first")

[<p class="inner-text first-item" id="first">

First paragraph.

</p>]

## Using CSS Selectors

You can also search for items using [CSS selectors](https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Getting_started/Selectors). These selectors are how the CSS language allows developers to specify HTML tags to style. Here are some examples:

p a – finds all a tags inside of a p tag.

body p a – finds all a tags inside of a p tag inside of a body tag.

html body – finds all body tags inside of an html tag.

p.outer-text – finds all p tags with a class of outer-text.

p#first – finds all p tags with an id of first.

body p.outer-text – finds any p tags with a class of outer-text inside of a body tag.

You can learn more about CSS selectors [here](https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Getting_started/Selectors).

BeautifulSoup objects support searching a page via CSS selectors using the select method. We can use CSS selectors to find all the p tags in our page that are inside of a div like this:

soup**.**select("div p")

[<p class="inner-text first-item" id="first">

First paragraph.

</p>, <p class="inner-text">

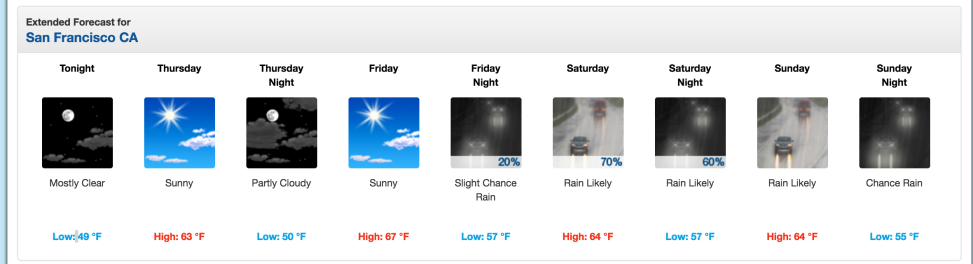
Second paragraph.

</p>]

Note that the select method above returns a list of BeautifulSoup objects, just like find and find\_all.

## Downloading weather data

We now know enough to proceed with extracting information about the local weather from the National Weather Service website. The first step is to find the page we want to scrape. We’ll extract weather information about downtown San Francisco from [this page](http://forecast.weather.gov/MapClick.php?lat=37.7772&lon=-122.4168).



We'll extract data about the extended forecast.

As you can see from the image, the page has information about the extended forecast for the next week, including time of day, temperature, and a brief description of the conditions.

## Exploring page structure with Chrome DevTools

We now know enough to download the page and start parsing it. In the below code, we:

* Download the web page containing the forecast.
* Create a BeautifulSoup class to parse the page.
* Find the div with id seven-day-forecast, and assign to seven\_day
* Inside seven\_day, find each individual forecast item.
* Extract and print the first forecast item.

page **=** requests**.**get("http://forecast.weather.gov/MapClick.php?lat=37.7772&lon=-122.4168")

soup **=** BeautifulSoup(page**.**content, 'html.parser')

seven\_day **=** soup**.**find(id**=**"seven-day-forecast")

forecast\_items **=** seven\_day**.**find\_all(class\_**=**"tombstone-container")

tonight **=** forecast\_items[0]

print(tonight**.**prettify())

<div class="tombstone-container">

<p class="period-name">

Tonight

<br>

<br/>

</br>

</p>

<p>

<img alt="Tonight: Mostly clear, with a low around 49. West northwest wind 12 to 17 mph decreasing to 6 to 11 mph after midnight. Winds could gust as high as 23 mph. " class="forecast-icon" src="newimages/medium/nfew.png" title="Tonight: Mostly clear, with a low around 49. West northwest wind 12 to 17 mph decreasing to 6 to 11 mph after midnight. Winds could gust as high as 23 mph. "/>

</p>

<p class="short-desc">

Mostly Clear

</p>

<p class="temp temp-low">

Low: 49 °F

</p>

</div>

## Extracting information from the page

As you can see, inside the forecast item tonight is all the information we want. There are 4 pieces of information we can extract:

* The name of the forecast item – in this case, Tonight.
* The description of the conditions – this is stored in the title property of img.
* A short description of the conditions – in this case, Mostly Clear.
* The temperature low – in this case, 49 degrees.

We’ll extract the name of the forecast item, the short description, and the temperature first, since they’re all similar:

period **=** tonight**.**find(class\_**=**"period-name")**.**get\_text()

short\_desc **=** tonight**.**find(class\_**=**"short-desc")**.**get\_text()

temp **=** tonight**.**find(class\_**=**"temp")**.**get\_text()

print(period)

print(short\_desc)

print(temp)

Tonight

Mostly Clear

Low: 49 °F

Now, we can extract the title attribute from the img tag. To do this, we just treat the BeautifulSoup object like a dictionary, and pass in the attribute we want as a key:

img **=** tonight**.**find("img")

desc **=** img['title']

print(desc)

Tonight: Mostly clear, with a low around 49. West northwest wind 12 to 17 mph decreasing to 6 to 11 mph after midnight. Winds could gust as high as 23 mph.

## Extracting all the information from the page

Now that we know how to extract each individual piece of information, we can combine our knowledge with css selectors and list comprehensions to extract everything at once.

In the below code, we:

* Select all items with the class period-name inside an item with the class tombstone-container in seven\_day.
* Use a list comprehension to call the get\_text method on each BeautifulSoup object.

period\_tags **=** seven\_day**.**select(".tombstone-container .period-name")

periods **=** [pt**.**get\_text() **for** pt **in** period\_tags]

periods

['Tonight',

'Thursday',

'ThursdayNight',

'Friday',

'FridayNight',

'Saturday',

'SaturdayNight',

'Sunday',

'SundayNight']

### css select 可以一次性搜索多层， 否则需要逐层搜索。

As you can see above, our technique gets us each of the period names, in order. We can apply the same technique to get the other 3 fields:

short\_descs **=** [sd**.**get\_text() **for** sd **in** seven\_day**.**select(".tombstone-container .short-desc")]

temps **=** [t**.**get\_text() **for** t **in** seven\_day**.**select(".tombstone-container .temp")]

descs **=** [d["title"] **for** d **in** seven\_day**.**select(".tombstone-container img")]

print(short\_descs)

print(temps)

print(descs)

['Mostly Clear', 'Sunny', 'Mostly Clear', 'Sunny', 'Slight ChanceRain', 'Rain Likely', 'Rain Likely', 'Rain Likely', 'Chance Rain']

['Low: 49 °F', 'High: 63 °F', 'Low: 50 °F', 'High: 67 °F', 'Low: 57 °F', 'High: 64 °F', 'Low: 57 °F', 'High: 64 °F', 'Low: 55 °F']

['Tonight: Mostly clear, with a low around 49. West northwest wind 12 to 17 mph decreasing to 6 to 11 mph after midnight. Winds could gust as high as 23 mph. ', 'Thursday: Sunny, with a high near 63. North wind 3 to 5 mph. ', 'Thursday Night: Mostly clear, with a low around 50. Light and variable wind becoming east southeast 5 to 8 mph after midnight. ', 'Friday: Sunny, with a high near 67. Southeast wind around 9 mph. ', 'Friday Night: A 20 percent chance of rain after 11pm. Partly cloudy, with a low around 57. South southeast wind 13 to 15 mph, with gusts as high as 20 mph. New precipitation amounts of less than a tenth of an inch possible. ', 'Saturday: Rain likely. Cloudy, with a high near 64. Chance of precipitation is 70%. New precipitation amounts between a quarter and half of an inch possible. ', 'Saturday Night: Rain likely. Cloudy, with a low around 57. Chance of precipitation is 60%.', 'Sunday: Rain likely. Cloudy, with a high near 64.', 'Sunday Night: A chance of rain. Mostly cloudy, with a low around 55.']

## Combining our data into a Pandas Dataframe ### 用来把网页扫描结果格式化输出

We can now combine the data into a [Pandas](http://pandas.pydata.org/) DataFrame and analyze it. A DataFrame is an object that can store tabular data, making data analysis easy. If you want to learn more about Pandas, check out our free to start course [here](https://www.dataquest.io/course/data-analysis-intermediate).

In order to do this, we’ll call the [DataFrame](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html) class, and pass in each list of items that we have. We pass them in as part of a dictionary. Each dictionary key will become a column in the DataFrame, and each list will become the values in the column:

weather **=** pd**.**DataFrame({

"period": periods,

"short\_desc": short\_descs,

"temp": temps,

"desc":descs

})

weather

|  | **desc** | **period** | **short\_desc** | **temp** |
| --- | --- | --- | --- | --- |
| **0** | Tonight: Mostly clear, with a low … | Tonight | Mostly Clear | Low: 49 °F |
| **1** | Thursday: Sunny, with a high near 63… | Thursday | Sunny | High: 63 °F |
| **2** | Thursday Night: Mostly clear, with a … | ThursdayNight | Mostly Clear | Low: 50 °F |
| **3** | Friday: Sunny, with a high near 67… | Friday | Sunny | High: 67 °F |
| **4** | Friday Night: A 20 percent chance of rain… | FridayNight | Slight ChanceRain | Low: 57 °F |
| **5** | Saturday: Rain likely. Cloudy, with a high.. | Saturday | Rain Likely | High: 64 °F |
| **6** | Saturday Night: Rain likely. Cloudy… | SaturdayNight | Rain Likely | Low: 57 °F |
| **7** | Sunday: Rain likely. Cloudy, with a … | Sunday | Rain Likely | High: 64 °F |
| **8** | Sunday Night: A chance of rain. Mostly… | SundayNight | Chance Rain | Low: 55 °F |

We can now do some analysis on the data. For example, we can use a regular expression and the [Series.str.extract](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.Series.str.extract.html) method to pull out the numeric temperature values:

temp\_nums **=** weather["temp"]**.**str**.**extract("(?P<temp\_num>\d+)", expand**=False**)

weather["temp\_num"] **=** temp\_nums**.**astype('int')

temp\_nums

0 49

1 63

2 50

3 67

4 57

5 64

6 57

7 64

8 55

Name: temp\_num, dtype: object

We could then find the mean of all the high and low temperatures:

weather["temp\_num"]**.**mean()

58.444444444444443

We could also only select the rows that happen at night:

is\_night **=** weather["temp"]**.**str**.**contains("Low")

weather["is\_night"] **=** is\_night

is\_night

0 True

1 False

2 True

3 False

4 True

5 False

6 True

7 False

8 True

Name: temp, dtype: bool

weather[is\_night]

|  | **desc** | **period** | **short\_desc** | **temp** | **temp\_num** | **is\_night** |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | Tonight: Mostly clear, with a low around 49. W... | Tonight | Mostly Clear | Low: 49 °F | 49 | True |
| **2** | Thursday Night: Mostly clear, with a low aroun... | ThursdayNight | Mostly Clear | Low: 50 °F | 50 | True |
| **4** | Friday Night: A 20 percent chance of rain afte... | FridayNight | Slight ChanceRain | Low: 57 °F | 57 | True |
| **6** | Saturday Night: Rain likely. Cloudy, with a l... | SaturdayNight | Rain Likely | Low: 57 °F | 57 | True |
| **8** | Sunday Night: A chance of rain. Mostly cloudy... | SundayNight | Chance Rain | Low: 55 °F | 55 | True |