11.6.3

Scraping Mars News

Robin will now try scraping the Mars news site. After collecting the data, she'll have the option to store it and later to analyze it.

In this section, we'll follow Robin as she scrapes the Mars news

(https://redplanetscience.com/) website.

NOTE

In this week's Challenge, you'll use the same process that you use in this section to scrape additional data from the Mars news site.

Write the Initial Code

To begin, create a new Jupyter notebook named `mars_news.ipynb`.

Next, we want to import the relevant libraries and modules. To do so, in the first cell, enter and run the following code:

```
# Import Splinter and BeautifulSoup
from splinter import Browser
from bs4 import BeautifulSoup as soup
from webdriver_manager.chrome import ChromeDriverManager
```

Then, we want to set the executable path. To do so, in the next cell, enter and run the following code:

```
executable_path = {'executable_path': ChromeDriverManager().install()}
browser = Browser('chrome', **executable_path, headless=False)
```

Next, we want to assign the URL, and instruct the browser to visit it. To do so, in the next cell, enter and run the following code:

```
# Visit the Mars NASA news site
url = 'https://redplanetscience.com'
browser.visit(url)
# Optional delay for loading the page
browser.is_element_present_by_css('div.list_text', wait_time=1)
```

In the preceding code, the last line, `browser.is_element_present_by_css('div.list_text', wait_time=1)`, accomplishes two things. The first is that the `is_element_present_by_css` method searches for elements with a specific combination of tag (`div`) and attribute (`list_text`). For example, if we used `ul.item_list`, the method would find `<ul class="item_list">`.

NOTE

Recall that a CSS `class` attribute gets represented with a period (`.`). And, an `id` attribute gets represented with a hashtag (`#`). So, a `div` element that has a `class` attribute of `list_text` gets notated as `div.list_text`. Likewise, a `button` element that has an `id` attribute of `more` gets notated as `button#more`. And, remember that an `id` attribute value must be unique on a webpage.

The second thing that the last line of code accomplishes is that it tells the browser to wait one second before searching for components. This optional delay is useful because sometimes, dynamic pages take a little while to load—especially if they have lots of images.

The last line of code returns `True`, indicating that it found a `div` element with a `list_text` attribute.

Now, practice using the `is_element_present_by_css` method yourself in the following Skill Drill:

SKILL DRILL

Use the `is_element_present_by_css` method to determine if `button#more` exists on the page. Does the method return `True` or `False`?

Next, we want to set up the HTML parser. To do so, in the next cell, enter and run the following code:

```
html = browser.html
news_soup = soup(html, 'html.parser')
slide_elem = news_soup.select_one('div.list_text')
```

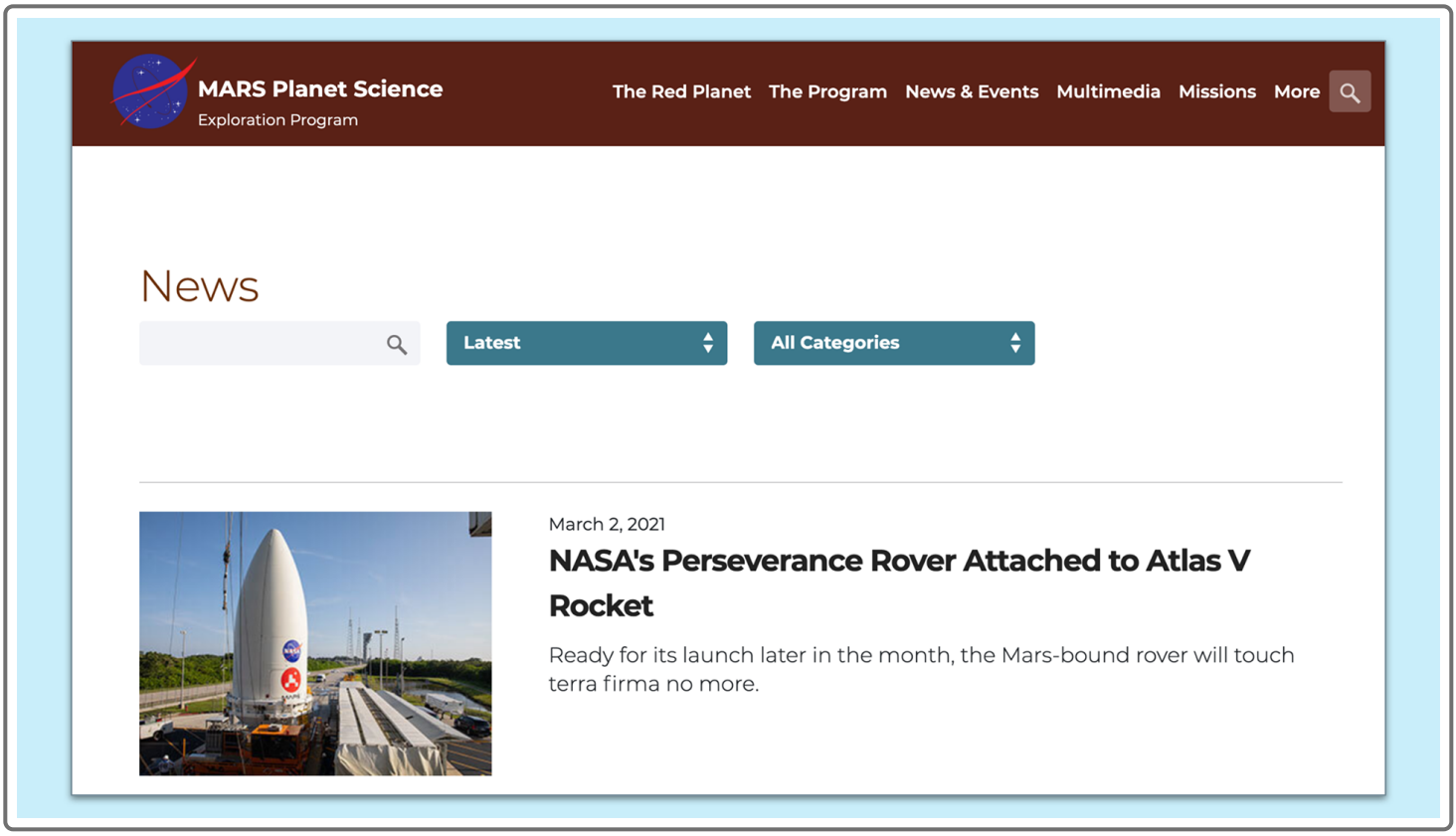
In the preceding code, we search for a `<div />` tag that has a `class` attribute of `list_text`, and we assign the result to the `slide_elem` variable. This result is the **parent element** of each article, which means that it holds all the other elements within it. Later, we'll reference this when we want to filter our search results further.

Scrape an Article Title and Its Summary Text

The data that Robin wants to extract is the title of the most-recent news article along with its summary text. Then, she'll be able to write a script to scrape the most-recent news articles.

To begin, navigate to the Mars news

(https://redplanetscience.com/) site. Notice that in the list of news articles, each article includes an image, publication date, title, and summary.



Next, right-click anywhere on the page, and then click Inspect. In DevTools, search for the HTML elements that you'll use to identify the title and the summary paragraph that you want.

Which HTML attribute will we use to scrape the article's title?

- ◯ `class = "article_teaser_body"`
- ◯ `div = "content_title"`
- ◯ `class = "content_title"`
- ◯ `id = "content_title"`

Check Answer

Finish ▶

We want to assign the title and the summary text to variables that we can reference later. To do so, begin scraping by entering and running the following code in the next cell:

```
title_elem = slide_elem.find('div', class_='content_title')
print(title_elem)
```

In the preceding code, we chained the `find` method to our previously assigned `slide_elem` variable. By doing so, we're saying, "This variable holds tons of information, so search that information to find this specific data." And, the specific data is the content title. We specify that inside the `find` method by using the `'div', class_='content_title'` parameters, which mean, "The specific data exists in a `<div />` element that has a `class` attribute of `content_title`."

The output of running the cell is the HTML code that contains the content title along with anything else that's nested inside that `<div />` element. Specifically, the output resembles the following:

```
<div class="content_title">Space History Is Made in This NASA Robot Facto
```

In the preceding output, notice that title appears within the HTML code, which is fantastic! But, we need just the title text. We don't need the extra HTML code. So, in the next cell, enter and run the following code:

```
title = title_elem.get_text()
print(title)
```

In the preceding code, the `get_text` method of the `title_elem` object returns only the title of the news article. So, the output consists of the most recently published title from the website. When the website gets updated with a new article, rerunning the code will return the title of that new article.

We have the title that we want, and that's a terrific start. Now, we need to get the summary text. To do so, we'll use a block of code that's similar to the last one.

What will we need to change in the following line of code to scrape the article summary instead of the title:

```
slide_elem.find("div", class_='content_title').get_text()
```

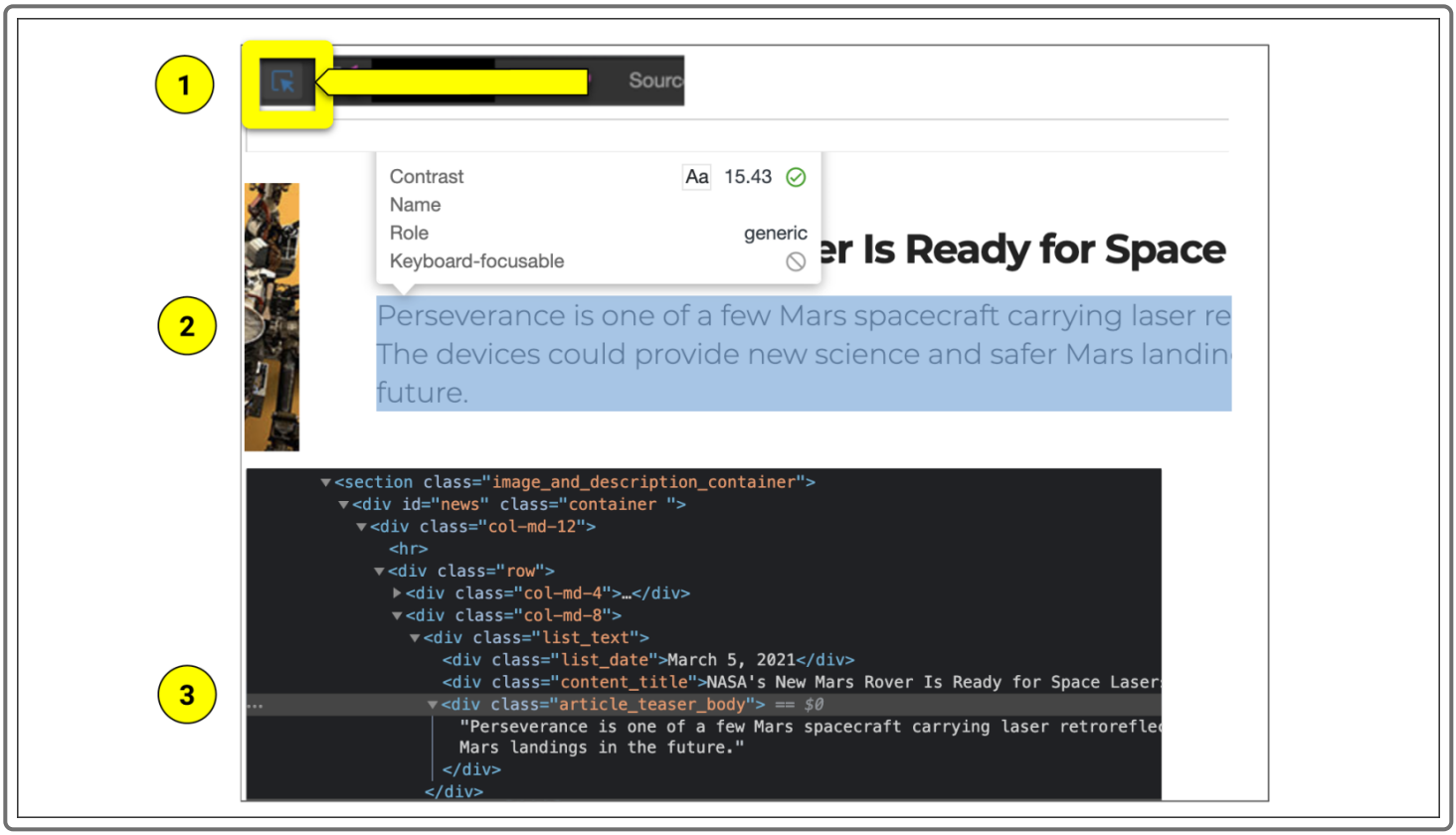
- ○ Nothing, we just need to filter through all of the other tags nested inside the first `<div />`.
- ○ We'll need to change the class to `"article_teaser_body."`
- ○ We'll need to switch from using a `"class"` to using an `"id"` because the summary and the title have different attributes.

Check Answer

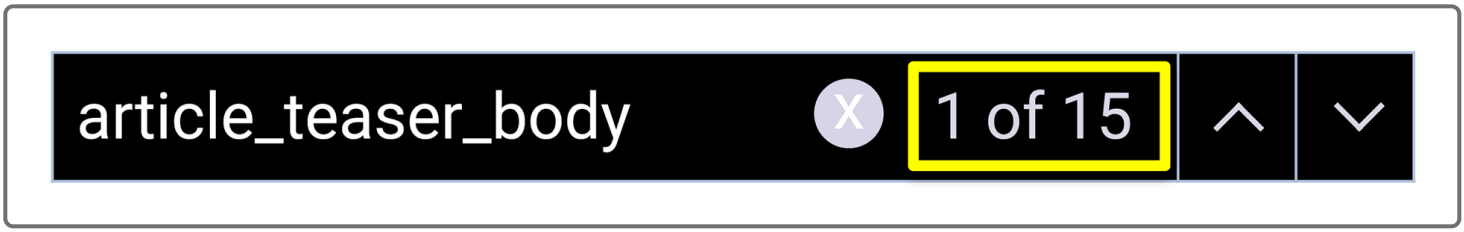
Finish

But before we add this code, we need to make sure that we scrape the right tag and class. To do so, in DevTools, click the "Select an element in the page to inspect it" button, and then on the webpage, select the article summary (also known as a **teaser**). Then in DevTools, check which tag is selected. It should be the following:

```
<div class="article_teaser_body">
```



We know that `article_teaser_body` is the right class name. In DevTools, search for `article_teaser_body`, and then notice that multiple matches exist. That's because multiple articles exist on the page, and each has a `<div />` tag with a `class` attribute of `article_teaser_body`.



For now, we want to extract only the first one—the most recent one. (We want only new news!) And because new articles get added to the beginning of the list, our search leads us to the first article.

To get the summary text, in the next cell, enter and run the following code:

```
# Use the parent element to find the paragraph text
news_p = slide_elem.find('div', class_='article_teaser_body').text
news_p
```

The output is just the article summary, which resembles that in the following image:

Example Summary Output

```
'NASA’s next mission to Mars, InSight, is scheduled to launch Saturday, May 5, on a first-ever mission to study the heart of the Red Planet.'
```

Finally, close the session. To do so, in the next cell, enter and run `browser.quit()`.

You learned lots of material in this lesson! You learned how to perform automated browser actions, like clicking a button. You learned how to scrape HTML tables by using Beautiful Soup and by using Pandas. Next, you’ll have the opportunity to use your new skills in this week’s Challenge assignment. Specifically, you’ll scrape data from multiple sources and then use your data analytic skills to wrangle, visualize, and analyze the data.