10.3.3 Scrape Mars Data: The News

After getting all of the tools set up and practicing all of our scraping skills, Robin's ready to start gathering data specific to her project, the Mission to Mars. She's really eager to create her application, and she has a list of websites she plans to use for scraping.

She's really excited about actually scraping live data. The script we're building is designed to scrape the most recent data—that means that each time we run the script, we'll pull the newest data available. As long as the website continues to be updated with new articles, which is likely, we'll have a constant influx of new information at our fingertips.

Robin has carefully staked out the websites she wants to routinely scrape data from. It's her dream to eventually work for NASA, so that's part of why she's chosen two similar sites as sources. Another reason is that these sites have very friendly Terms of Service (or ToS, also known as Terms of Use) when it comes to web scraping.

Many websites don't want automated browsers visiting their sites and snagging data. If there are too many visits, the server hosting the site could get overloaded and shut down. Administrators can then ban the IP address of the person doing the scraping, making it more difficult to even manually visit the site to view data.

IMPORTANT

> Terms of Service and Terms of Use bring up an ethical issue when gathering data. Many websites don't allow automated browsing and scraping—some of the scraping scripts out there are designed to gather data quickly, and the constant traffic can overload web servers and disable a website.

With those precautions out of the way, Robin's ready to start scraping Mars news.

In Jupyter Notebook, navigate back to your home folder and create a new `.ipynb` file. This time, name it "Mission_to_Mars" and begin with importing Splinter and BeautifulSoup in the first cell.

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

Remember, set your executable path in the next cell, then set up the URL (**NASA Mars News (https://redplanetscience.com)**) for scraping.

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

In the next cell of your Jupyter notebook, we'll assign the url and instruct the browser to visit it.

```
# 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)
```

With the following line, `browser.is_element_present_by_css('div.list_text', wait_time=1)`, we are accomplishing two things.

One is that we're searching for elements with a specific combination of tag (`div`) and attribute (`list_text`). As an example, `ul.item_list` would be found in HTML as `<ul class="item_list">`.

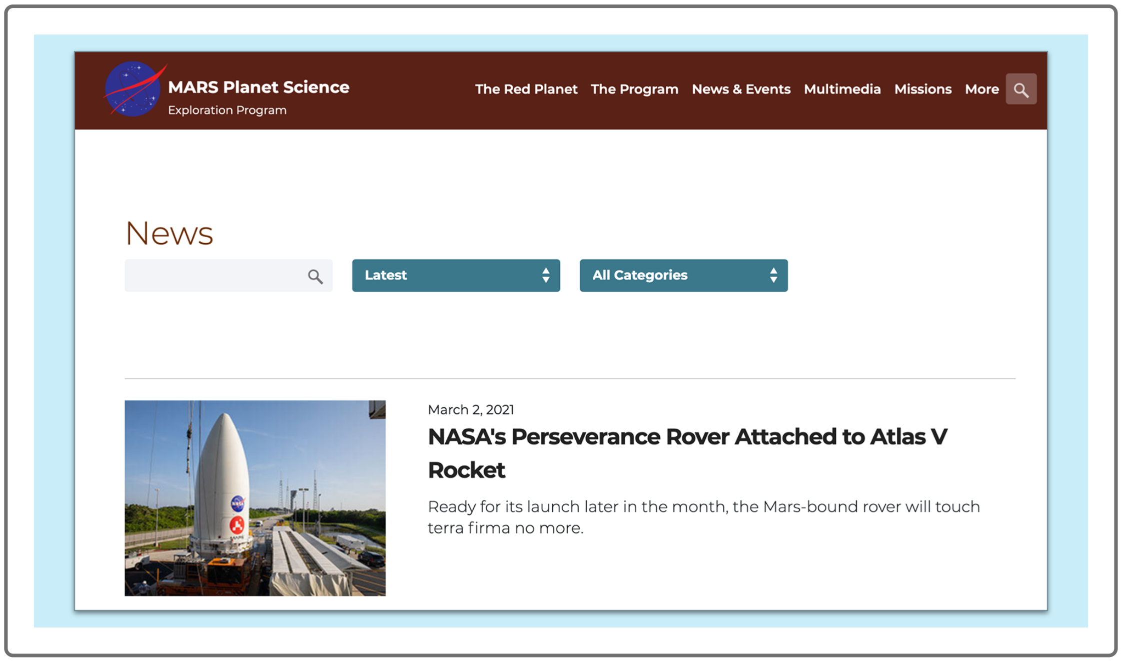
Secondly, we're also telling our browser to wait one second before searching for components. The optional delay is useful because sometimes dynamic pages take a little while to load, especially if they are image-heavy.

In the next empty cell, we'll set up the HTML parser:

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

Notice how we've assigned `slide_elem` as the variable to look for the `<div />` tag and its descendent (the other tags within the `<div />` element)? This is our parent element. This means that this element holds all of the other elements within it, and we'll reference it when we want to filter search results even further. The `.` is used for selecting classes, such as `list_text`, so the code `'div.list_text'` pinpoints the `<div />` tag with the class of `list_text`. CSS works from right to left, such as returning the last item on the list instead of the first. Because of this, when using `select_one`, the first matching element returned will be a `<li />` element with a class of `slide` and all nested elements within it.

The data Robin wants to collect from this particular website is the most recent news article along with its summary. Remember, the code for this will eventually be used in an application that will scrape live data with the click of a button—this site is dynamic and the articles will change frequently, which is why Robin is removing the manual task of retrieving each new article.



March 2, 2021

NASA's Perseverance Rover Attached to Atlas V Rocket

Ready for its launch later in the month, the Mars-bound rover will touch terra firma no more.

After opening the page in a new browser, right-click to inspect and activate your DevTools. Then search for the HTML components you'll use to identify the title and paragraph you want.

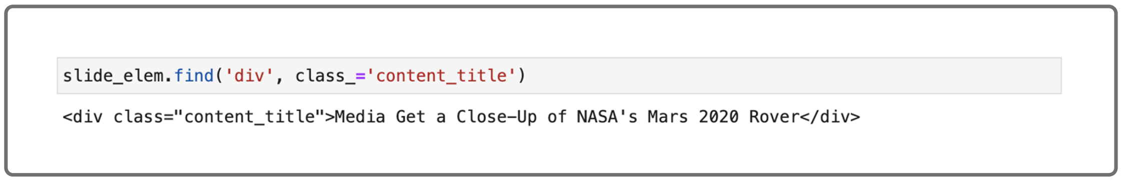
Retake

We'll want to assign the title and summary text to variables we'll reference later. In the next empty cell, let's begin our scraping. Type the following:

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

In this line of code, we chained `.find` onto our previously assigned variable, `slide_elem.` When we do this, we're saying, "This variable holds a ton of information, so look inside of that information to find *this* specific data." The data we're looking for is the content title, which we've specified by saying, "The specific data is in a `<div />` with a class of `'content_title'`."

Go ahead and run this cell. The output should be the HTML containing the content title and anything else nested inside of that `<div />`.



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

```
<div class="content_title">Media Get a Close-Up of NASA's Mars 2020 Rover</div>
```

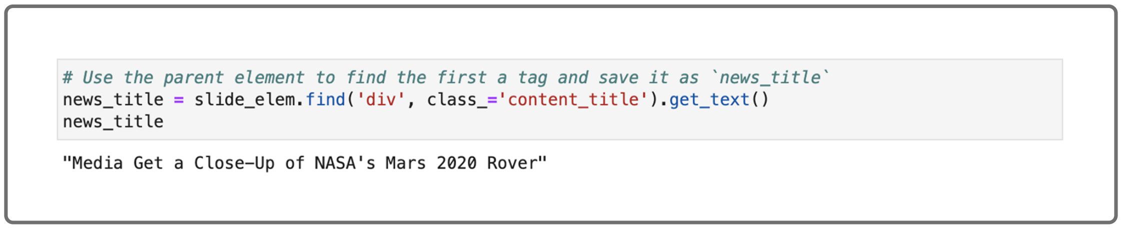
The title is in that mix of HTML in our output—that's awesome! But we need to get just the text, and the extra HTML stuff isn't necessary. In the next cell, type the following:

```
# Use the parent element to find the first `a` tag and save it as `news_titl
news_title = slide_elem.find('div', class_='content_title').get_text()
news_title
```

We've added something new to our `.find()` method here: `.get_text()`. When this new method is chained onto `.find()`, only the text of the element is returned. The code above, for example, would return only the title of the news article and not any of the HTML tags or elements.

Retake

Once executed, the result is the most recent title published on the website. When the website is updated and a new article is posted, when our code is run again, it will return that article instead.



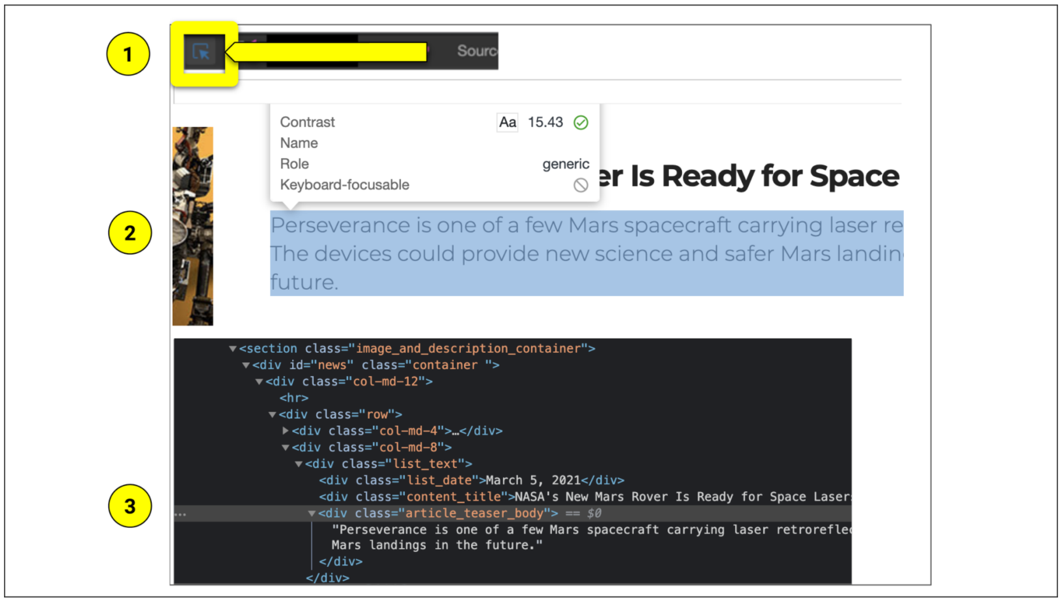
```
# Use the parent element to find the first a tag and save it as `news_title`
news_title = slide_elem.find('div', class_='content_title').get_text()
news_title

"Media Get a Close-Up of NASA's Mars 2020 Rover"
```

If you aren't seeing this exact title, that's fine, because the webpage has likely been updated since the screenshot was taken. If you're not seeing a title at all, then check to make sure that you have selected the correct tag and attribute (such as `div` and `class`) in your `find()` function.

We have the title we want, and that's a great start. Next we need to add the summary text. This next block of code will be very similar to our last one.

Retake

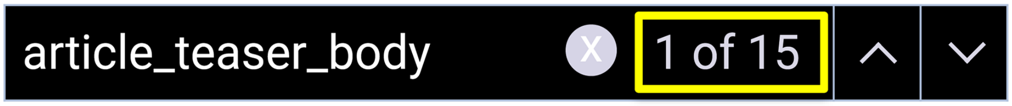
Before we can update our code, we'll need to use our DevTools to make sure we're scraping the right tag and class. Use the DevTools selector tool and select the article summary (teaser), then check to see which tag is highlighted.



Perseverance is one of a few Mars spacecraft carrying laser re
The devices could provide new science and safer Mars landin
future.

```
▼<section class="image_and_description_container">
  ▼<div id="news" class="container ">
    ▼<div class="col-md-12">
       <hr>
      ▼<div class="row">
        ►<div class="col-md-4">…</div>
        ▼<div class="col-md-8">
          ▼<div class="list_text">
             <div class="list_date">March 5, 2021</div>
             <div class="content_title">NASA's New Mars Rover Is Ready for Space Laser
...         ▼<div class="article_teaser_body"> == $0
               "Perseverance is one of a few Mars spacecraft carrying laser retrorefle
               Mars landings in the future."
             </div>
           </div>
```

We know that `"article_teaser_body"` is the right class name, but when we search for it, there is more than one result. What now?



That's okay. There will be many matches because there are many articles, each with a tag of `<div />` and a class of `article_teaser_body`. We want to pull the first one on the list, not a specific one, so more than 10 results is fine. In this case, if our scraping code is too specific, we'd pull only that article summary instead of the most recent.

Because new articles are added to the top of the list, and we only need the most recent one, our search leads us to the first article. *New* news only, please!

IMPORTANT

> There are two methods used to find tags and attributes with BeautifulSoup:
>
> - `.find()` is used when we want only the first class and attribute we've specified.
> - `.find_all()` is used when we want to retrieve *all* of the tags and attributes.
>
> For example, if we were to use `.find_all()` instead of `.find()` when pulling the summary, we would retrieve all of the summaries on the page instead of just the first one.

In the next empty cell in your Jupyter Notebook, type the following:

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

When you run this cell, your output should only be the summary of the article.

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.'
```