10.3.4 Scrape Mars Data: Featured Image

Robin has finished her first Mars-related scrape. All of the practice and study is starting to pay off. It's a fantastic first step to creating her web application. The next step is to scrape the featured image from another Mars website. Once the image is scraped, we'll want to add it to our web app as well.

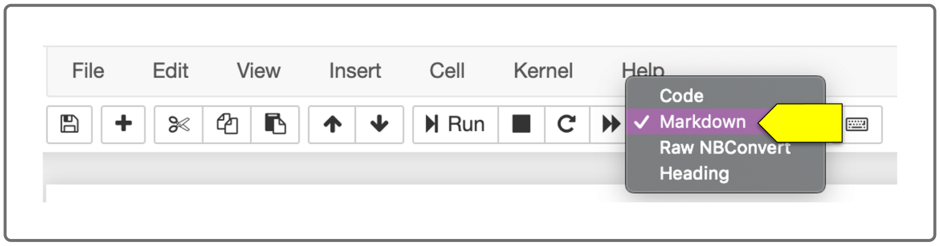
Ultimately, with each item we scrape, we'll also save and then serve it on our own website. We're basically using pieces from other websites to piece together our own website, with news and images custom tailored to Robin's taste.

She knows that adding a finished, functional web application to her portfolio will go a long way toward helping her achieve her goal of working for NASA.

Robin's next step scraping code will be to gather the featured images from the Jet Propulsion Laboratory's **Space Images** (https://spaceimages-mars.com) webpage. In your Jupyter notebook, use markdown to separate the article scraping from the image scraping.

In the next empty cell, type `### Featured Images` and change the format of the code cell to `"Markdown."`

You can access the cell formatting feature by using a drop-down menu at the top of the notebook. It's currently set to "Code," so click the down arrow to toggle the drop-down menu and select "Markdown" instead.



The cell below this one is where we'll begin our scraping. First, let's check out the webpage.

The first image that pops up on the webpage is the featured image. Robin wants the full-size version of this image, so we know we'll want Splinter to click the "Full Image" button. From there, the page directs us to a slideshow. It's a little closer to getting the full-size feature image, but we aren't quite there yet.

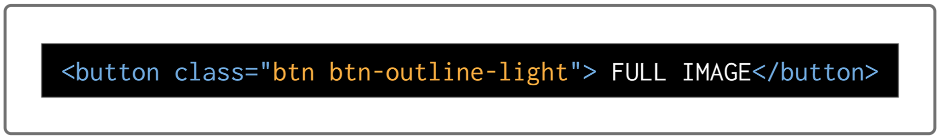
This is a lot of clicking to get to the image we want. Let's start getting our code ready to automate all of the clicks.

In the next notebook cell, set up the URL:

```
# Visit URL
url = 'https://spaceimages-mars.com'
browser.visit(url)
```

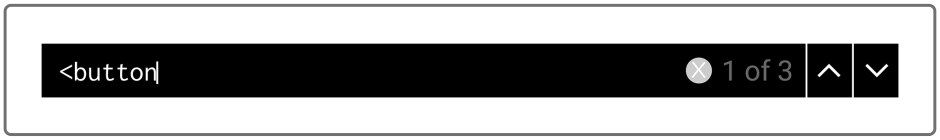
Run this code to make sure it's working correctly. A new automated browser should open to the featured images webpage.

Next, we want to click the "Full Image" button. This button will direct our browser to an image slideshow. Let's take a look at the button's HTML tags and attributes with the DevTools.



```
<button class="btn btn-outline-light"> FULL IMAGE</button>
```

This is a fairly straightforward HTML tag: the `<button>` element has two classes (`btn` and `btn-outline-light`) and a string reading "FULL IMAGE". First, let's use the dev tools to search for all the `button` elements.



<button

Since there are only three buttons, and we want to click the full-size image button, we can go ahead and use the HTML tag in our code.

In the next cell, let's type the following:

```
# Find and click the full image button
full_image_elem = browser.find_by_tag('button')[1]
full_image_elem.click()
```

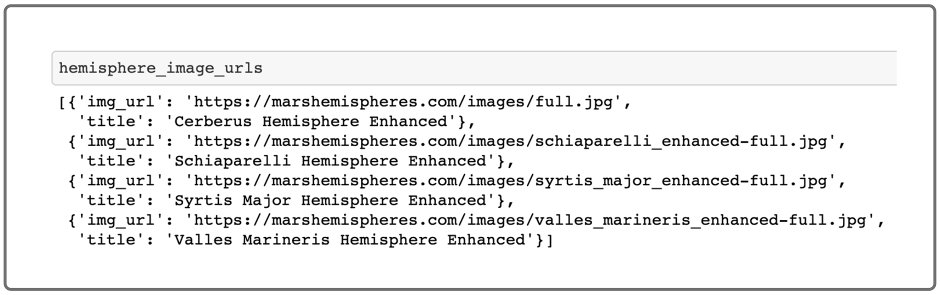
Notice the indexing chained at the end of the first line of code? With this, we've stipulated that we want our browser to click the second button.

Go ahead and run this code. The automated browser should automatically "click" the button and change the view to a slideshow of images, so we're on the right track. We need to click the More Info button to get to the next page. Let's look at the DevTools again to see what elements we can use for our scraping.

With the new page loaded onto our automated browser, it needs to be parsed so we can continue and scrape the full-size image URL. In the next empty cell, type the following:

```
# Parse the resulting html with soup
html = browser.html
img_soup = soup(html, 'html.parser')
```

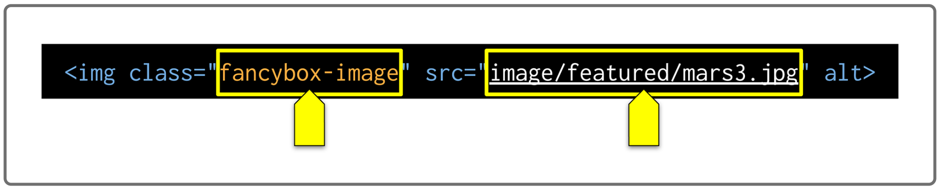
Now we need to find the relative image URL. In our browser (make sure you're on the same page as the automated one), activate your DevTools again. This time, let's find the image link for that image. This is a little more tricky. Remember, Robin wants to pull the most recently posted image for her web app. If she uses the image URL below, she'll only ever pull that specific image when using her app.



```
hemisphere_image_urls
```

```
[{'img_url': 'https://marshemispheres.com/images/full.jpg',
  'title': 'Cerberus Hemisphere Enhanced'},
 {'img_url': 'https://marshemispheres.com/images/schiaparelli_enhanced-full.jpg',
  'title': 'Schiaparelli Hemisphere Enhanced'},
 {'img_url': 'https://marshemispheres.com/images/syrtis_major_enhanced-full.jpg',
  'title': 'Syrtis Major Hemisphere Enhanced'},
 {'img_url': 'https://marshemispheres.com/images/valles_marineris_enhanced-full.jpg',
  'title': 'Valles Marineris Hemisphere Enhanced'}]
```

It's important to note that the value of the **src** will be different every time the page is updated, so we can't simply record the current value—we would only pull that image each time the code is executed, instead of the most recent one.



```
<img class="fancybox-image" src="image/featured/mars3.jpg" alt>
```

We'll use the image tag and class (`<img />`and `fancybox-img`) to build the URL to the full-size image. Let's go back to Jupyter Notebook to do that.

```
# Find the relative image url
img_url_rel = img_soup.find('img', class_='fancybox-image').get('src')
img_url_rel
```

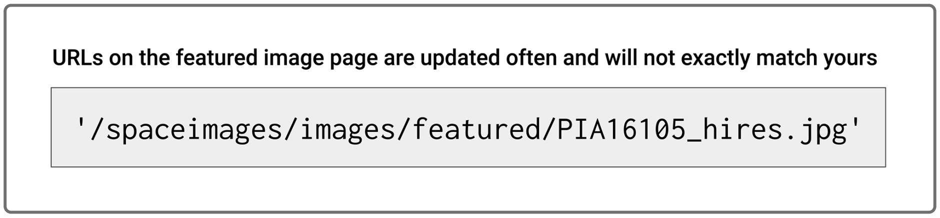
We've done a lot with that single line.

Let's break it down:

- An `img` tag is nested within this HTML, so we've included it.
- `.get('src')` pulls the link to the image.

What we've done here is tell BeautifulSoup to look inside the `<img />` tag for an image with a class of `fancybox-image`. Basically we're saying, "This is where the image we want lives—use the link that's inside these tags."

Run the notebook cell to see the output of the link.

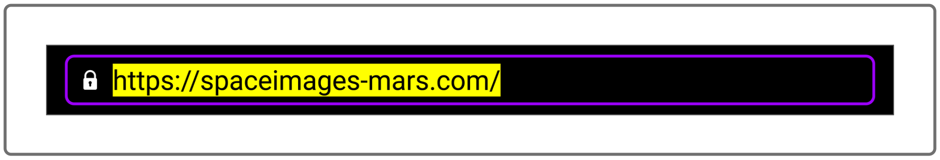


URLs on the featured image page are updated often and will not exactly match yours

```
'/spaceimages/images/featured/PIA16105_hires.jpg'
```

This looks great! We were able to pull the link to the image by pointing BeautifulSoup to where the image will be, instead of grabbing the URL directly. This way, when JPL updates its image page, our code will still pull the most recent image.

But if we copy and paste this link into a browser, it won't work. This is because it's only a partial link, as the base URL isn't included. If we look at our address bar in the webpage, we can see the entire URL up there already; we just need to add the first portion to our app.



https://spaceimages-mars.com/

Let's add the base URL to our code.

```
# Use the base URL to create an absolute URL
img_url = f'https://spaceimages-mars.com/{img_url_rel}'
img_url
```

We're using an f-string for this print statement because it's a cleaner way to create print statements; they're also evaluated at run-time. This means that it, and the variable it holds, doesn't exist until the code is executed and the values are not constant. This works well for our scraping app because the data we're scraping is live and will be updated frequently.