Chapter 3:
A Practical Introduction to Web Scraping in Python

DR. LINDA MAHMOUDI

”يرفعِ اللهُ الذين أمنوا منكم والذين أوتوا العلم درجات“

In this chapter, we'll learn how to:

- Parse website data using **string methods** and **regular expressions**
- Parse website data using an **HTML parser**
- Interact with **forms** and other website components

Scrape and Parse Text From Websites

- uses Python's built-in IDLE editor
- to create and edit Python files and interact with the Python shell
- use a page that's hosted on Real Python's server.
- we'll start grabbing all the HTML code from a single web page.

What Is Python IDLE?

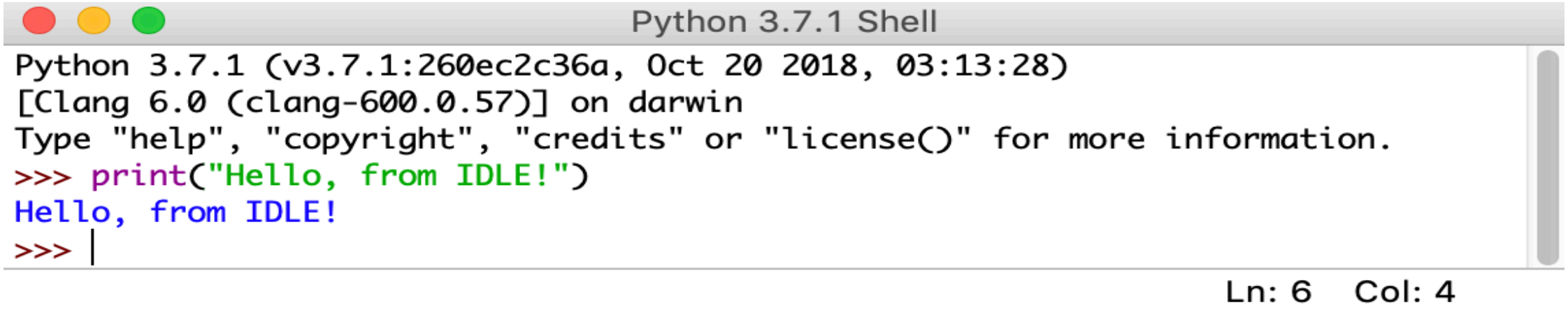
- Every Python installation comes with an Integrated Development and Learning Environment (IDLE or even IDE). These are a class of applications that help us write code more efficiently.
- Python IDLE is very bare-bones, which makes it the perfect tool for a beginning programmer.
- The best place to experiment with Python code is in the interactive interpreter, otherwise known as a **shell**. The shell is a basic Read-Eval-Print Loop (REPL).

What Is Python IDLE?

- It reads a Python statement, evaluates the result of that statement, and then prints the result on the screen. Then, it loops back to read the next statement.
- Every programmer needs to be able to edit and save text files. Python programs are files with the **.py** extension that contain lines of Python code.
- Python IDLE gives you the ability to create and edit these files with ease.

How to Use the Python IDLE Shell ?

The shell is the default mode of operation for Python IDLE. When we click on the icon to open the program, the shell is the first thing that we see:



This is a blank Python interpreter window. we can use it to start interacting with Python immediately.

Build Your First Web Scraper

- One useful package for web scraping that we can find in Python's standard library is **urllib**,
- which contains tools for working with URLs.
- In particular, the **urllib.request** module contains a function called **urlopen()** that you can use to open a URL within a program.
- In **IDLE's** interactive window, type the following to import **urlopen()**:

```python
>>> from urllib.request import urlopen
```

- **The web page that you'll open is at the following URL:**

```python
>>> url = "http://olympus.realpython.org/profiles/aphrodite"
```

- **To open the web page, pass url to urlopen():**

```python
>>> page = urlopen(url)
```

- To extract the HTML from the page, first use the HTTPResponse object's **.read()** method, which returns a sequence of bytes. Then use **.decode()** to decode the bytes to a string using UTF-8:

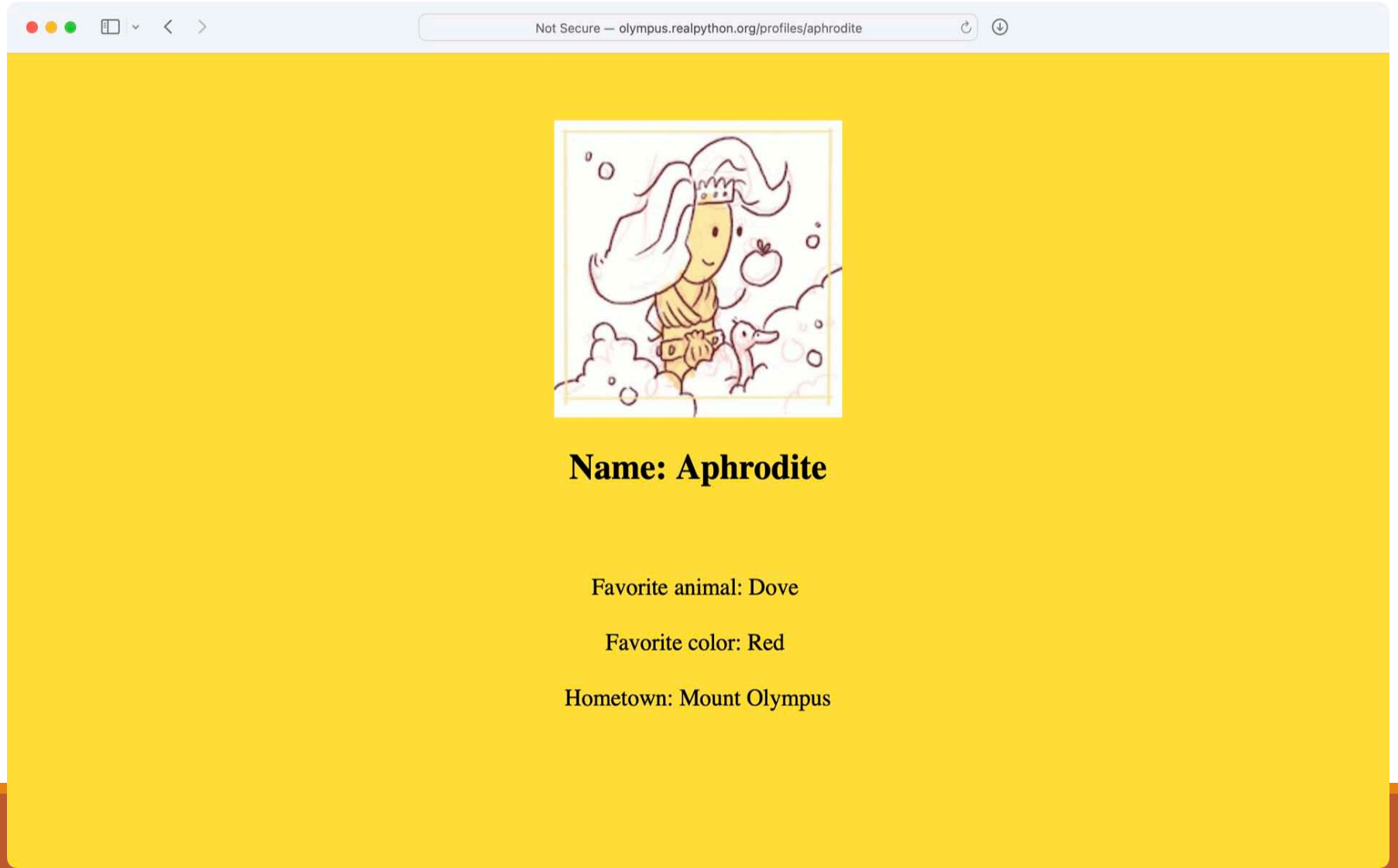
```python
>>> html_bytes = page.read()
>>> html = html_bytes.decode("utf-8")
```

- Now you can print the HTML to see the contents of the web page:

Python

```
>>> print(html)
<html>
<head>
<title>Profile: Aphrodite</title>
</head>
<body bgcolor="yellow">
<center>
<br><br>
<img src="/static/aphrodite.gif" />
<h2>Name: Aphrodite</h2>
<br><br>
Favorite animal: Dove
<br><br>
Favorite color: Red
<br><br>
Hometown: Mount Olympus
</center>
</body>
</html>
```

- The output that we're seeing is the HTML code of the website, which the browser renders when we visit http://olympus.realpython.org/profiles/aphrodite:



Extract Text From HTML With String Methods

- One way to extract information from a web page's HTML is to use string methods. For instance, we can use **.find()** to search through the text of the HTML for the <title> tags and extract the title of the web page.

- To start, we'll extract the title of the web page that you requested in the previous example. If you know the index of the first character of the title and the index of the first character of the closing </title> tag, then you can use a string **slice** to extract the title.

❖Because **.find()** returns the index of the first occurrence of a substring, we can get the index of the opening <title> tag by passing the string "<title>" to .find():

```
Python
>>> title_index = html.find("<title>")
>>> title_index
14
```

❖You don’t want the index of the <title> tag, though. You want the index of the title itself. To get the index of the first letter in the title, you can add the length of the string "<title>" to title_index:

```
Python
>>> start_index = title_index + len("<title>")
>>> start_index
21
```

- Now get the index of the closing </title> tag by passing the string "</title>" to .find():

```
Python

>>> end_index = html.find("</title>")
>>> end_index
39
```

- Finally, you can extract the title by slicing the html string:

```
Python

>>> title = html[start_index:end_index]
>>> title
'Profile: Aphrodite'
```

- Real-world HTML can be much more complicated and far less predictable than the HTML on the Aphrodite profile page. Here’s another profile page with some messier HTML that you can scrape:

Python

```
>>> url = "http://olympus.realpython.org/profiles/poseidon"
>>> page = urlopen(url)
>>> html = page.read().decode("utf-8")
>>> start_index = html.find("<title>") + len("<title>")
>>> end_index = html.find("</title>")
>>> title = html[start_index:end_index]
>>> title
'\n<head>\n<title >Profile: Poseidon'
```

Whoops! There's a bit of HTML mixed in with the title. Why's that?

The HTML for the /profiles/poseidon page looks similar to the /profiles/aphrodite page, but there's a small difference. The opening <title> tag has an extra space before the closing angle bracket (>), rendering it as <title >.

❖ **These sorts of problems can occur in countless unpredictable ways.**

we need a more reliable way to extract text from HTML.

✓ Get to Know Regular Expressions

Regular expressions—or regexes for short—are patterns that you can use to search for text within a string

within a string. Python supports regular expressions through the standard library's `re` module.

> **Note:** Regular expressions aren't particular to Python. They're a general programming concept and are supported in many programming languages.

To work with regular expressions, the first thing that you need to do is import the `re` module:

Python >>>

```
>>> import re
```

Regular expressions use special characters called **metacharacters** to denote different patterns. For instance, the asterisk character (`*`) stands for zero or more instances of whatever comes just before the asterisk.

In the following example, you use `.findall()` to find any text within a string that matches a given regular expression:

Python >>>

```
>>> re.findall("ab*c", "ac")
['ac']
```

Exercice: Scrape data from a website

Write a program that grabs the full HTML from the following URL:

Python >>>

```
>>> url = "http://olympus.realpython.org/profiles/dionysus"
```

Then use `.find()` to display the text following *Name:* and *Favorite Color:* (not including any leading spaces or trailing HTML tags that might appear on the same line).

Use an HTML Parser for Web Scraping in Python

- Although regular expressions are great for pattern matching in general, sometimes it's easier to use an HTML parser that's explicitly designed for parsing out HTML pages.
- There are many Python tools written for this purpose, but the Beautiful Soup library is a good one to start with.
- To Create a BeautifulSoup Object:
  - Type the following program into a new editor window:

```python
# beauty_soup.py

from bs4 import BeautifulSoup
from urllib.request import urlopen

url = "http://olympus.realpython.org/profiles/dionysus"
page = urlopen(url)
html = page.read().decode("utf-8")
soup = BeautifulSoup(html, "html.parser")
```

This program does three things:

1.Opens the URL http://olympus.realpython.org/profiles/dionysus by using urlopen() from the urllib.request module

2.Reads the HTML from the page as a string and assigns it to the html variable

3.Creates a BeautifulSoup object and assigns it to the soup variable

Use a BeautifulSoup Object

Python

```
>>> print(soup.get_text())


Profile: Dionysus




Name: Dionysus

Hometown: Mount Olympus

Favorite animal: Leopard

Favorite Color: Wine
```

Python

```
>>> soup.find_all("img")
[<img src="/static/dionysus.jpg"/>, <img src="/static/grapes.png"/>]
```

Python

```
>>> image1, image2 = soup.find_all("img")
```

Python

```
>>> image1.name
'img'
```

Python

```
>>> image1["src"]
'/static/dionysus.jpg'

>>> image2["src"]
'/static/grapes.png'
```

Python

```
>>> soup.title
<title>Profile: Dionysus</title>
```

Name: Dionysus

Elements | Console | Sources | Network | Timelines | Storage | Graphics | Layers | Audit

dionysus 〉 Response

Breakpoints

All Exceptions

Uncaught Exceptions

Assertion Failures

By Type | By Path

dionysus — olympus.realpython.org

Filter | All

```
1  <html>
2  <head>
3  <TITLE >Profile: Dionysus</title  / >
4  </head>
5  <body bgcolor="yellow">
6  <center>
7  <br><br>
8  <img src="/static/dionysus.jpg" />
9  <h2>Name: Dionysus</h2>
10 <img src="/static/grapes.png"><br><br>
11 Hometown: Mount Olympus
12 <br><br>
13 Favorite animal: Leopard <br>
14 <br>
```

You can also retrieve just the string between the title tags with the .string property of the Tag object:

Python

```
>>> soup.title.string
'Profile: Dionysus'
```

Python

```
>>> soup.find_all("img", src="/static/dionysus.jpg")
[<img src="/static/dionysus.jpg"/>]
```

Exercice: Parse HTML with Beautiful Soup

Write a program that grabs the full HTML from the page at the URL http://olympus.realpython.org/profiles.

Using Beautiful Soup, print out a list of all the links on the page by looking for HTML tags with the name a and retrieving the value taken on by the href attribute of each tag.

The final output should look like this:

Shell

```
http://olympus.realpython.org/profiles/aphrodite
http://olympus.realpython.org/profiles/poseidon
http://olympus.realpython.org/profiles/dionysus
```

Make sure that you only have one slash (/) between the base URL and the relative URL.

Interact With HTML Forms

- The urllib module that we've been working with so far in this tutorial is well suited for requesting the contents of a web page.
- we need to interact with a web page to obtain the content we need.
- For example, we might need to submit a form or click a button to display hidden content.

Interact With HTML Forms

- The Python standard library doesn’t provide a built-in means for working with web pages interactively, but many third-party packages are available from PyPI. Among these, **MechanicalSoup** is a popular and relatively straightforward package to use.
- In essence, MechanicalSoup installs what’s known as a **headless browser**, which is a web browser with no graphical user interface. This browser is controlled programmatically via a Python program.

Install MechanicalSoup

```shell
$ python -m pip install MechanicalSoup
```

Create a Browser Object

```python
>>> import mechanicalsoup
>>> browser = mechanicalsoup.Browser()
```

Browser objects represent the headless web browser. We can use them to request a page from the Internet by passing a URL to their .get() method:

```python
>>> url = "http://olympus.realpython.org/login"
>>> page = browser.get(url)
```

page is a Response object that stores the response from requesting the URL from the browser:

```python
>>> page
<Response [200]>
```

The number 200 represents the status code returned by the request. A status code of 200 means that the request was successful. An unsuccessful request might show a status code of 404 if the URL doesn’t exist or 500 if there’s a server error when making the request.

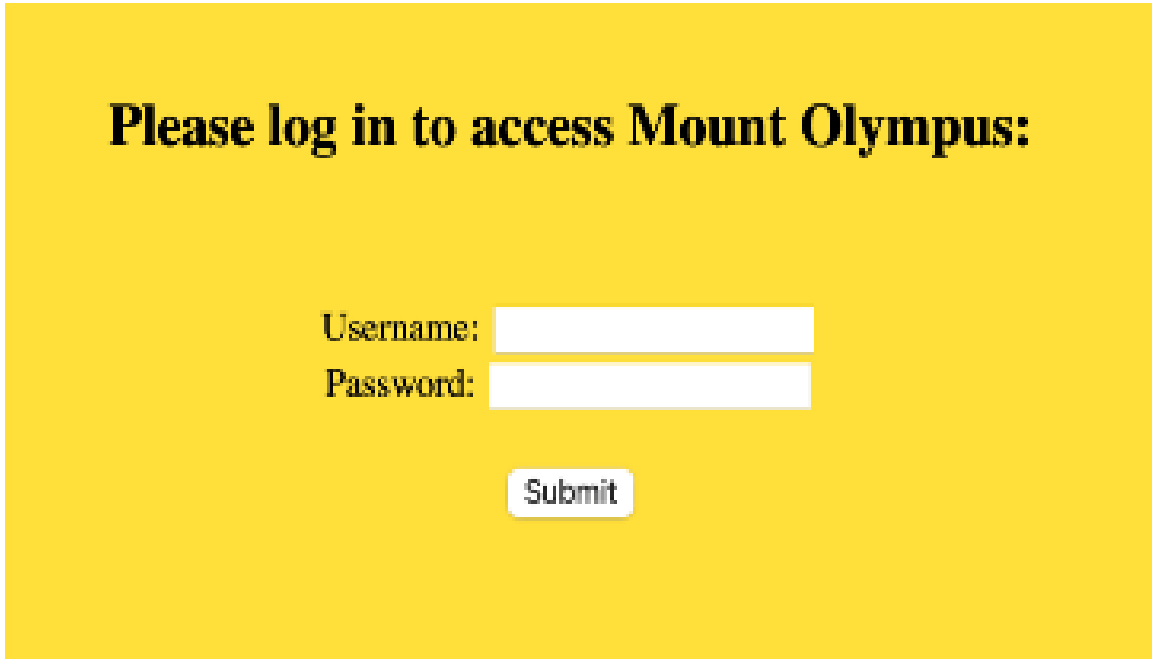
MechanicalSoup uses Beautiful Soup to parse the HTML from the request, and page has a .soup attribute that represents a BeautifulSoup object:

```
Python
>>> type(page.soup)
<class 'bs4.BeautifulSoup'>
```

we can view the HTML by inspecting the .soup attribute:

```
>>> page.soup
<html>
<head>
<title>Log In</title>
</head>
<body bgcolor="yellow">
<center>
<br/><br/>
<h2>Please log in to access Mount Olympus:</h2>
<br/><br/>
<form action="/login" method="post" name="login">
Username: <input name="user" type="text"/><br/>
Password: <input name="pwd" type="password"/><br/><br/>
<input type="submit" value="Submit"/>
</form>
</center>
</body>
</html>
```

Submit a Form With MechanicalSoup



Notice this page has a <form> on it with <input> elements for a username and a password.

- Try typing in a random username and password combination.
- If we guess incorrectly, then the message Wrong username or password! is displayed at the bottom of the page.
- However, if we provide the correct login credentials, then we're redirected to the /profiles page.

Username	Password
zeus	ThunderDude

- we’ll see how to use MechanicalSoup to fill out and submit this form using Python!
- The important section of HTML code is the login form—that is, everything inside the <form> tags.
- The <form> on this page has the name attribute set to login.
- This form contains two <input> elements, one named user and the other named pwd.
- The third <input> element is the Submit button.

Python

```python
import mechanicalsoup

# 1
browser = mechanicalsoup.Browser()
url = "http://olympus.realpython.org/login"
login_page = browser.get(url)
login_html = login_page.soup

# 2
form = login_html.select("form")[0]
form.select("input")[0]["value"] = "zeus"
form.select("input")[1]["value"] = "ThunderDude"

# 3
profiles_page = browser.submit(form, login_page.url)
```

- To confirm that we've successfully logged in, type the following into the interactive window:

```
Python

>>> profiles_page.url
'http://olympus.realpython.org/profiles'
```

  - We confirm now that the submission successfully redirected to the /profiles page. If something had gone wrong, then the value of profiles_page.url would still be "http://olympus.realpython.org/login".

- we create a Browser instance and use it to request the URL http://olympus.realpython.org/login. You assign the HTML content of the page to the login_html variable using the .soup property.

- login_html.select("form") returns a list of all <form> elements on the page. Because the page has only one <form> element, we can access the form by retrieving the element at index 0 of the list. When there is only one form on a page, we may also use login_html.form.
  - The next two lines select the username and password inputs and set their value to "zeus" and "ThunderDude", respectively.
- You submit the form with browser.submit(). Notice that you pass two arguments to this method, the form object and the URL of the login_page, which you access via login_page.url

- Now that we have the profiles_page variable set, it’s time to programmatically obtain the URL for each link on the /profiles page.
- To do this, we use .select() again, this time passing the string "a" to select all the <a> anchor elements on the page:

```python
>>> links = profiles_page.soup.select("a")
```

Now you can iterate over each link and print the href attribute:

Python

```
>>> for link in links:
...     address = link["href"]
...     text = link.text
...     print(f"{text}: {address}")
...
Aphrodite: /profiles/aphrodite
Poseidon: /profiles/poseidon
Dionysus: /profiles/dionysus
```

Exercice: Submit a form with MechanicalSoup

Use MechanicalSoup to provide the correct username (zeus) and password (ThunderDude) to the login form located at the URL http://olympus.realpython.org/login.

Once the form is submitted, display the title of the current page to determine that you've been redirected to the /profiles page.

Your program should print the text <title>All Profiles</title>.