urllib.parse

The **urllib.parse** library is your standard interface for breaking up URL strings and combining them back together. You can use it to convert a relative URL to an absolute URL, for example. Let's try using it to parse a URL that includes a query:

```
from urllib.parse import urlparse
result = urlparse('https://duckduckgo.com/?q=python+stubbing&t=canonical&ia=qa')
print (result)
#ParseResult(scheme='https', netloc='duckduckgo.com', path='/', params='', query='q=python+st

print (result.netloc)
#'duckduckgo.com'

print (result.geturl())
#'https://duckduckgo.com/?q=python+stubbing&t=canonical&ia=qa'

print (result.port)
#None
```

Here we import the **urlparse** function and pass it an URL that contains a search query to the duckduckgo website. My query was to look up articles on "python stubbing". As you can see, it returned a **ParseResult** object that you can use to learn more about the URL. For example, you can get the port information (None in this case), the network location, path and much more.

Submitting a Web Form

This module also holds the **urlencode** method, which is great for passing data to a URL. A typical use case for the urllib.parse library is submitting a web form. Let's find out how you might do that by having the duckduckgo search engine look for Python:

```
import urllib.request
import urllib.parse
data = urllib.parse.urlencode({'q': 'Python'})
data

#'q=Python'

url = 'http://duckduckgo.com/html/'
full_url = url + '?' + data
response = urllib.request.urlopen(full_url)
with open('results.html', 'wb') as f:
    f.write(response.read())
```

This is pretty straightforward. Basically we want to submit a query to duckduckgo ourselves using Python instead of a browser. To do that, we need to construct our query string using **urlencode**. Then we put that together to create a fully qualified URL and use urllib.request to submit the form. We then grab the result and save it to disk.