Crawling the Web

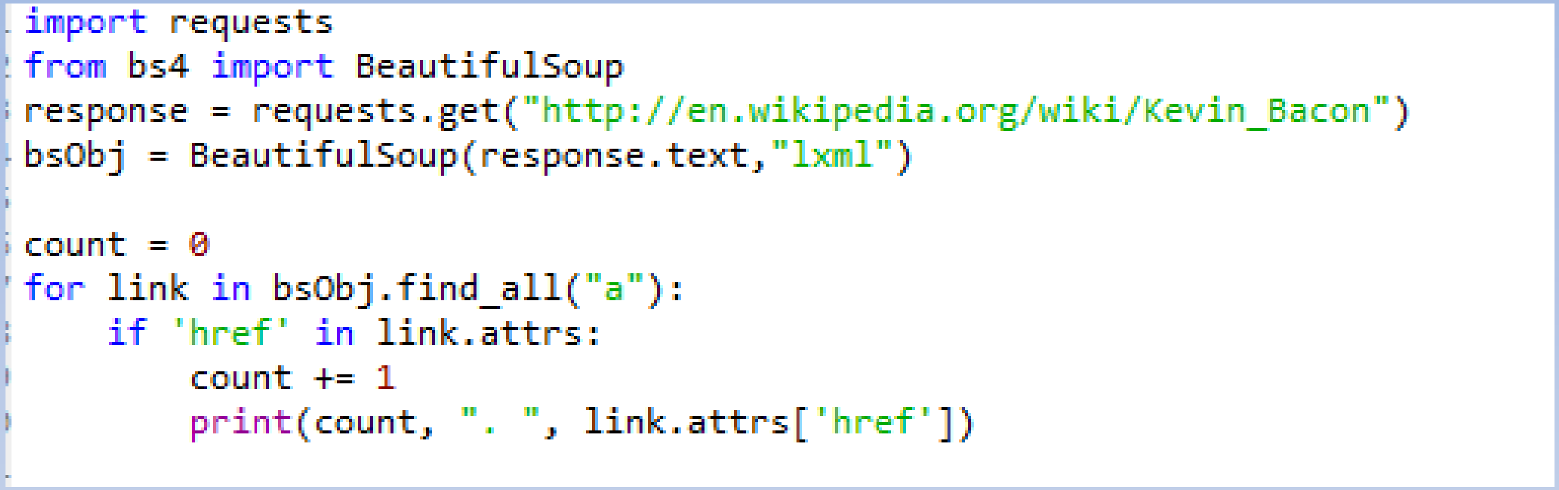
- Advanced Scrapers travel multiple pages or even multiple sites
- Web crawlers? Crawling across the web!
  - Recursion is a core element of the operation
    - Retrieve page contents for a URL
    - Examine that page for another URL
    - Retrieve *that* page, and so on …
- Just because crawling the web is easy, it doesn't mean that you should do it!

Extracting Links

- Hyperlinks are typically attributes under the anchor tag (<a></a>)
- Extract all the links out and decide which ones you want to keep
- Accessing Attributes
  - Very useful to do in tags like <a> when you want to get the URL it's pointing to under "href", or in <img> if you want to get the target image under "src"
  - List of attributes are obtained using myTag.attrs
  - Each attribute in turn is a dictionary object

```
import requests
from bs4 import BeautifulSoup
```

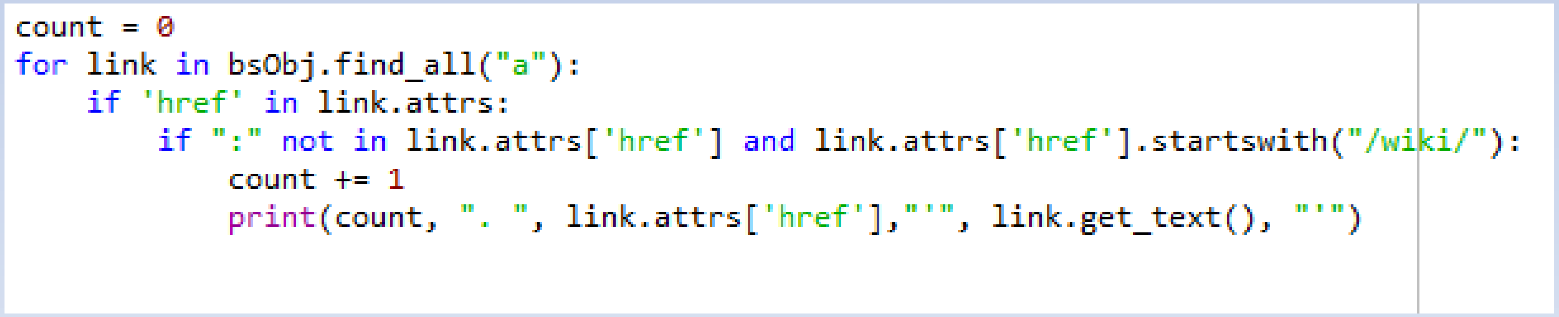
Extracting Links



```python
import requests
from bs4 import BeautifulSoup
response = requests.get("http://en.wikipedia.org/wiki/Kevin_Bacon")
bsObj = BeautifulSoup(response.text,"lxml")

count = 0
for link in bsObj.find_all("a"):
    if 'href' in link.attrs:
        count += 1
        print(count, ". ", link.attrs['href'])
```

- Refine your extracted data further

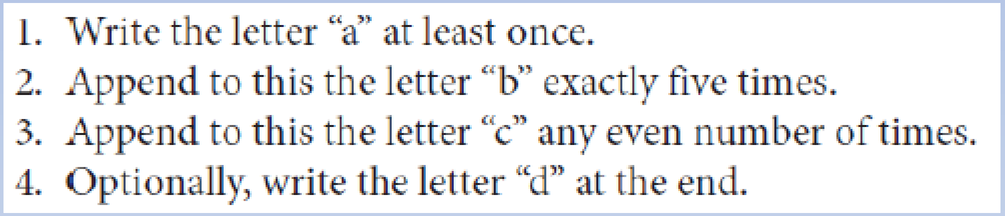


```python
count = 0
for link in bsObj.find_all("a"):
    if 'href' in link.attrs:
        if ":" not in link.attrs['href'] and link.attrs['href'].startswith("/wiki/"):
            count += 1
            print(count, ". ", link.attrs['href'],"'", link.get_text(), "'")
```

- If you want to use more advanced techniques for finding patterns, use *Regular Expressions*

Regular Expressions

- They are used to identify “regular string”
  - Given a set of rules what’s returned are strings that match it
  - A “regular string” is any string that can be generated by a series of linear rules



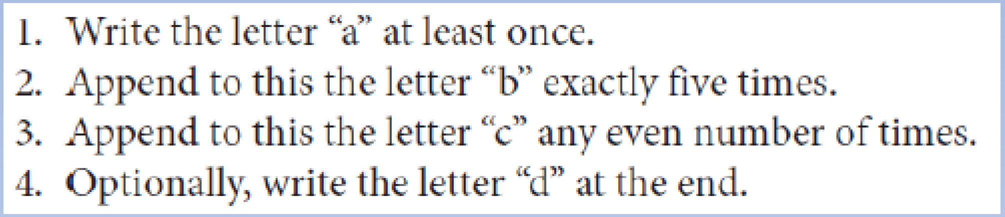
1. Write the letter “a” at least once.
2. Append to this the letter “b” exactly five times.
3. Append to this the letter “c” any even number of times.
4. Optionally, write the letter “d” at the end.

- In other words, the program can say:
  - “Yes, this string you’ve given me follows the rules and so I will return it”

  OR

  - “This string does not follow the rules, and I will discard it”

Regular Expressions


1. Write the letter "a" at least once.
2. Append to this the letter "b" exactly five times.
3. Append to this the letter "c" any even number of times.
4. Optionally, write the letter "d" at the end.

- Strings that follow these rules are:

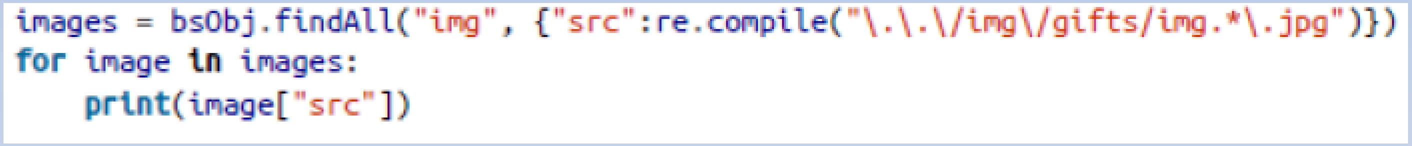
  "aaaabbbbbcccc", "aabbbbbcc", and so on - there are an infinite number of variations

- Regular expressions are a shorthand way of expressing these set of rules:

  a*bbbbb(cc)*(d|)

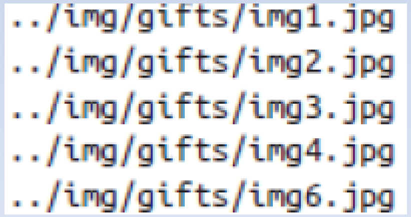
Regular Expressions

- Say we are interested in grabbing URLs of all product images on a web site
  - They are in a particular folder
  - They all start with a certain name and end in ".jpg"



```
images = bsObj.findAll("img", {"src":re.compile("\.\.\/img\/gifts/img.*\.jpg")})
for image in images:
    print(image["src"])
```

  - It will print out only the relative paths that start with ../img/gifts/img and end in .jpg



```
../img/gifts/img1.jpg
../img/gifts/img2.jpg
../img/gifts/img3.jpg
../img/gifts/img4.jpg
../img/gifts/img6.jpg
```

Crawling across the Internet

- Before you start writing a program that crawls across web sites
  - What data are you trying to gather?
    - Scrape a few predetermined websites or try to discover new websites?
  - On a particular website
    - Do you want to immediately follow the next outbound link to a new website
    - Do you want to stick around for a while and drill down into the current website?
  - Do you want to exclude some websites based on conditions (natural languages?)
  - How do you protect yourself against legal action if your web crawler catches somebody's attention on a website?