STAT679
Computing for Data Science and Statistics

Lecture 12: Structured Data from the Web

Lots of interesting data resides on websites

HTML : **H**yper**T**ext **M**arkup **L**anguage
Specifies basically everything you see on the Internet

XML : E**X**tensible **M**arkup **L**anguage
Designed to be an easier way for storing data, similar framework to HTML

JSON : **J**ava**S**cript **O**bject **N**otation
Designed to be a saner version of XML

SQL : **S**tructured **Q**uery **L**anguage
IBM-designed language for interacting with databases

APIs : **A**pplication **P**rogramming **I**nterface
Allow interaction with website functionality (e.g., Google maps)

Three Aspects of Data on the Web

Location: URL (Uniform Resource Locator), IP address

Specifies location of a computer on a network

Protocol: HTTP, HTTPS, FTP, SMTP

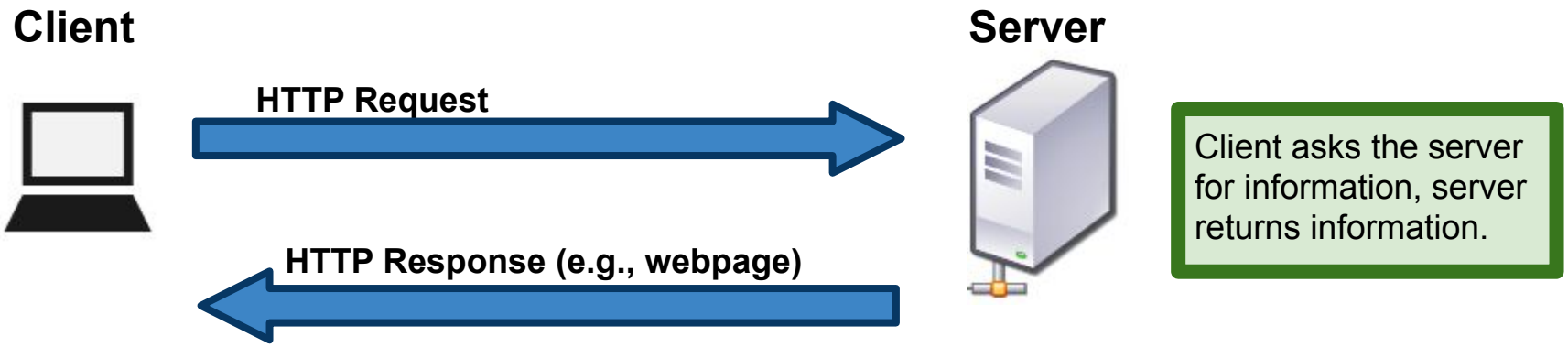
Specifies how computers on a network should communicate with one another

Content: HTML, JSON, XML (for example)

Contains actual information, e.g., tells browser what to display and how

We'll mostly be concerned with website content. Wikipedia has good entries on network protocols. The classic textbook is *Computer Networks* by A. S. Tanenbaum.

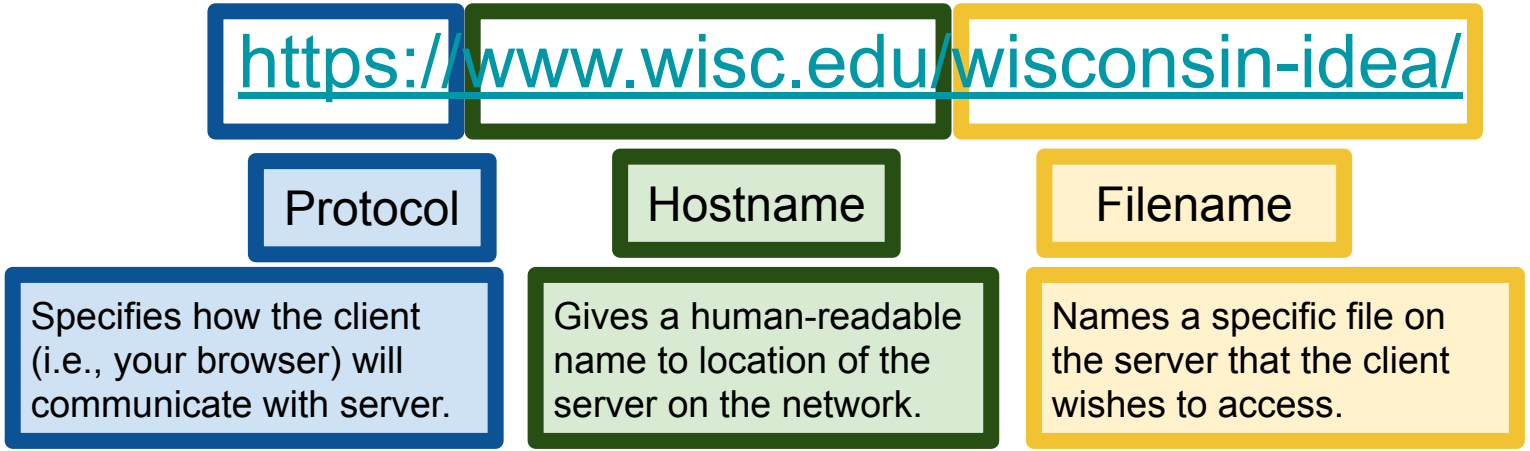
Client-server model



HTTP is

- **Connectionless:** after a request is made, the client disconnects and waits
- **Stateless:** server and client “forget about each other” after a request
- **Media agnostic:** any kind of data can be sent over HTTP

Anatomy of a URL



Note: often the extension of the file will indicate what type it is (e.g., html, txt, pdf, etc), but not always. Often, one must determine the type of the file based on its contents. This can almost always be done automatically.

Accessing websites in Python: `urllib`

Python library for opening URLs and interacting with websites

https://docs.python.org/3/howto/urllib2.html

Software development community is moving towards **requests**

https://requests.readthedocs.io/en/master/

a bit over-powered for what we want to do, but feel free to use it in HWs

Note: Python 3 split what was previously `urllib2` in Python 2 into several related submodules of `urllib`. You should be aware of this in case you end up having to migrate code from Python 2 to Python 3 or vice-versa.

Using `urllib`

`urllib.request.urlopen()` : opens the given url, returns a file-like object

```
1 import urllib.request
2 response = urllib.request.urlopen('http://pages.stat.wisc.edu/~kdlevin')
3 response
```

```
<http.client.HTTPResponse at 0x105f82a90>
```

Three basic methods

- `getcode()` : return the HTTP status code of the response
- `geturl()` : return URL of the resource retrieved (e.g., see if redirected)
- `info()` : return meta-information from the page, such as headers

getcode()

HTTP includes success/error status codes

Ex: 200 OK, 301 Moved Permanently, 404 Not Found, 503 Service Unavailable

See https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

```
1 import urllib.request
2 response = urllib.request.urlopen('http://pages.stat.wisc.edu/~kdlevin')
3 response.getcode()
```

```
200
```

```
1 response = urllib.request.urlopen('http://pages.stat.wisc.edu/~kdlevin/nonsense.html')
```

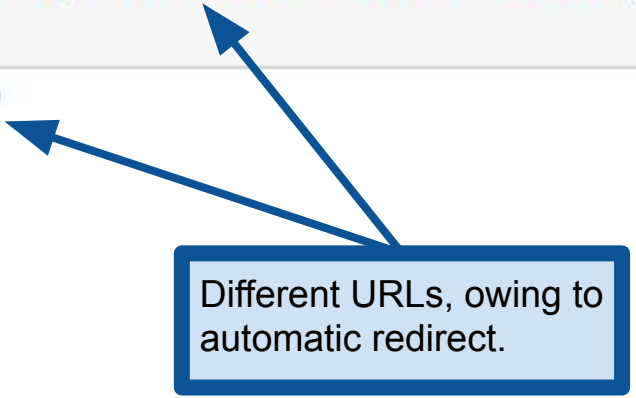
```
---------------------------------------------------------------------------
HTTPError                                 Traceback (most recent call last)
<ipython-input-14-00c8212845f4> in <module>()
----> 1 response = urllib.request.urlopen('http://pages.stat.wisc.edu

HTTPError: HTTP Error 404: Not Found
```

Note: I cropped a bunch of error information, which will normally be useful!

geturl()

```
1 response = urllib.request.urlopen('http://umich.edu/~klevin')
2 response.geturl()
```

```
'http://www-personal.umich.edu/~klevin/'
```



https://en.wikipedia.org/wiki/URL_redirection

`info()`

Returns a dictionary-like object with information about the page you retrieved.

```
1 response = urllib.request.urlopen('http://pages.stat.wisc.edu/~kdlevin')
2 print(response.info())
```

```
Date: Thu, 11 Mar 2021 03:24:41 GMT
Server: Apache
Last-Modified: Sat, 30 Jan 2021 08:16:54 GMT
ETag: "659d0474-1c3b-5ba19be58e980"
Accept-Ranges: bytes
Content-Length: 7227
Connection: close
Content-Type: text/html
```

This can be useful when you aren't sure of content type or character set used by a website, though nowadays most of those things are handled automatically by parsers.

HTML Crash Course

HTML is a markup language.

```
<tag_name attr1=”value” attr2=”differentValue”>String contents</tag_name>
```

Basic unit: **tag**

(usually) a start and end tag, like `<p>contents</p>`

Contents of a tag may contain more tags:

```
<head><title>The Title</title></head>
<p>This tag links to <a href=”google.com”>Google</a></p>
```

HTML Crash Course

```
<tag_name attr1="value" attr2="differentValue">String contents</tag_name>
```

Tags have attributes, which are specified after the tag name, in (key,value) pairs of the form `key="val"`

Example: hyperlink tags

```
<a href="pages.stat.wisc.edu/~kdlevin">My webpage</a>
```

Corresponds to a link to My personal webpage.

The `href` attribute specifies where the hyperlink should point.

HTML Crash Course: Recap

Of special interest in your homework: HTML tables

https://developer.mozilla.org/en-US/docs/Web/HTML/Element/table

https://www.w3schools.com/html/html_tables.asp

https://www.w3.org/TR/html401/struct/tables.html

Okay, back to `urllib`

`urllib` reads a webpage (full of HTML) and returns a “response” object

The response object can be treated like a file:

```
1 import urllib.request
2 response = urllib.request.urlopen('https://wikipedia.org')
3 response.read()
```

```
b'<!DOCTYPE html>\n<html lang="mul" class="no-js">\n<head>\n<meta charset="utf-8">\n<title>Wikipe
ame="description" content="Wikipedia is a free online encyclopedia, created and edited by volunte
and hosted by the Wikimedia Foundation.">\n<![if gt IE 7]>\n<script>\ndocument.documentElement.cl
ocumentElement.className.replace( /(^|\\s)no-js(\\s|$)/, "$1js-enabled$2" );\n</script>\n<![endif
><meta http-equiv="imagetoolbar" content="no"><![endif]-->\n<meta name="viewport" content="initia
ble=yes">\n<link rel="apple-touch-icon" href="/static/apple-touch/wikipedia.png">\n<link rel="sho
tatic/favicon/wikipedia.ico">\n<link rel="license" href="//creativecommons.org/licenses/by-sa/3.0
```

Okay, back to `urllib`

`urllib` reads a webpage (full of HTML) and returns a “response” object

The response object can be treated like a file:

```
1 import urllib.request
2 response = urllib.request.urlopen('https://wikipedia.org')
3 response.read()
```

```
b'<!DOCTYPE html>\n<html lang="mul" class="no-js">\n<head>\n<meta charset="utf-8">\n<title>Wikipe
ame="descripti                                                                    ed by volunte
and hosted by                                                                     entElement.cl
ocumentElement                                                                    pt>\n<![endif
><meta http-equiv="imagetoolbar" content="no"><![endif]-->\n<meta name="viewport" content="initia
ble=yes">\n<link rel="apple-touch-icon" href="/static/apple-touch/wikipedia.png">\n<link rel="sho
tatic/favicon/wikipedia.ico">\n<link rel="license" href="//creativecommons.org/licenses/by-sa/3.0
```

What a mess! How am I supposed to do anything with this?!

Parsing HTML/XML in Python: `beautifulsoup`

Python library for working with HTML/XML data

- Builds nice tree representation of markup data…
- ...and provides tools for working with that tree

Documentation: https://www.crummy.com/software/BeautifulSoup/bs4/doc/

Good tutorial:
http://www.pythonforbeginners.com/python-on-the-web/beautifulsoup-4-python/

Installation: `pip install beautifulsoup` or follow instructions for conda or...

Parsing HTML/XML in Python: beautifulsoup

BeautifulSoup turns HTML mess into a (sometimes complex) tree

Four basic kinds of objects:

Tag: corresponds to HTML tags

`<[name] [attr]=”xyz”>[string]</[name]>`)

Two important attributes: tag.name, tag.string

Also has dictionary-like structure for accessing attributes

NavigableString: special kind of string for use in `bs4`

BeautifulSoup: represents the HTML document itself

Comment: special kind of NavigableString for HTML comments

Example (from the BeautifulSoup docs)

```
html_doc = """
<html><head><title>The Dormouse's story</title></head>
<body>
<p class="title"><b>The Dormouse's story</b></p>

<p class="story">Once upon a time there were three little sisters; and their names were
<a href="http://example.com/elsie" class="sister" id="link1">Elsie</a>,
<a href="http://example.com/lacie" class="sister" id="link2">Lacie</a> and
<a href="http://example.com/tillie" class="sister" id="link3">Tillie</a>;
and they lived at the bottom of a well.</p>

<p class="story">...</p>
"""
from bs4 import BeautifulSoup
parsed = BeautifulSoup(html_doc, 'html.parser')
```

Follow along at home: https://www.crummy.com/software/BeautifulSoup/bs4/doc/#quick-start

```
1 print(parsed.prettify())
```

```
<html>
 <head>
  <title>
   The Dormouse's story
  </title>
 </head>
 <body>
  <p class="title">
   <b>
    The Dormouse's story
   </b>
  </p>
  <p class="story">
   Once upon a time there were three little sisters; and their names were
   <a class="sister" href="http://example.com/elsie" id="link1">
    Elsie
   </a>
   ,
   <a class="sister" href="http://example.com/lacie" id="link2">
    Lacie
   </a>
   and
   <a class="sister" href="http://example.com/tillie" id="link3">
    Tillie
   </a>
   ;
and they lived at the bottom of a well.
  </p>
  <p class="story">
```

BeautifulSoup supports “pretty printing” of HTML documents.

BeautifulSoup allows navigation of the HTML tags

```
1 parsed.title
```

```
<title>The Dormouse's story</title>
```

```
1 parsed.title.name
```

```
u'title'
```

```
1 parsed.title.string
```

```
u"The Dormouse's story"
```

```
1 parsed.find_all('a')
```

```
[<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>,
 <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>,
 <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>]
```

```
1 for link in parsed.find_all('a'):
2     print link.get('href')
```

```
http://example.com/elsie
http://example.com/lacie
http://example.com/tillie
```

Finds all the tags that have the name `'a'`, which is the HTML tag for a link.

The `'href'` attribute in a tag with name `'a'` contains the actual url for use in the link.

A note on attributes

HTML attributes and Python attributes are **different things!**

But in `BeautifulSoup` they collide in a weird way

`BeautifulSoup` tags have their HTML attributes accessible like a dictionary:

```
shortdoc="""
<p class="story">Once upon a time there were three little sisters; and their names were
<a href="http://example.com/elsie" class="sister" id="link1">Elsie</a>,
<a href="http://example.com/lacie" class="sister" id="link2">Lacie</a> and
<a href="http://example.com/tillie" class="sister" id="link3">Tillie</a>;
and they lived at the bottom of a well.</p>
"""
pshort = BeautifulSoup(shortdoc, 'html.parser')
print pshort.p['class']
```

```
[u'story']
```

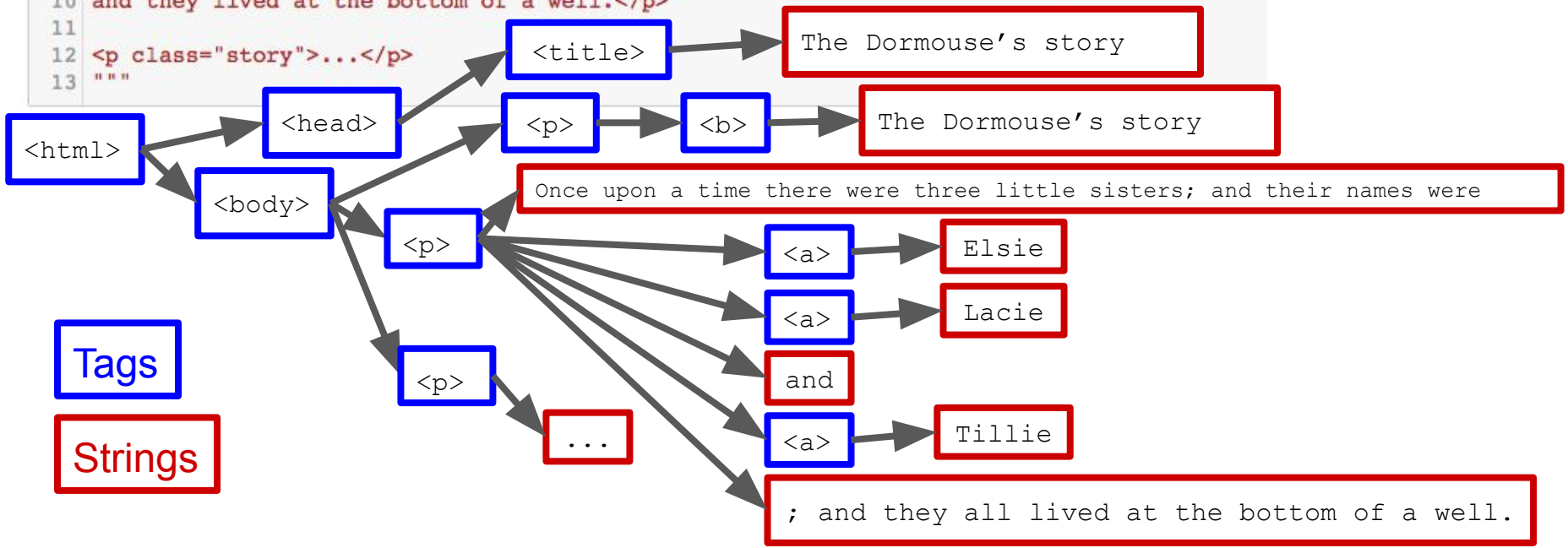
`BeautifulSoup` tags have their *children* accessible as Python attributes:

```
print pshort.p.a
```

```
<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>
```

HTML tree structure

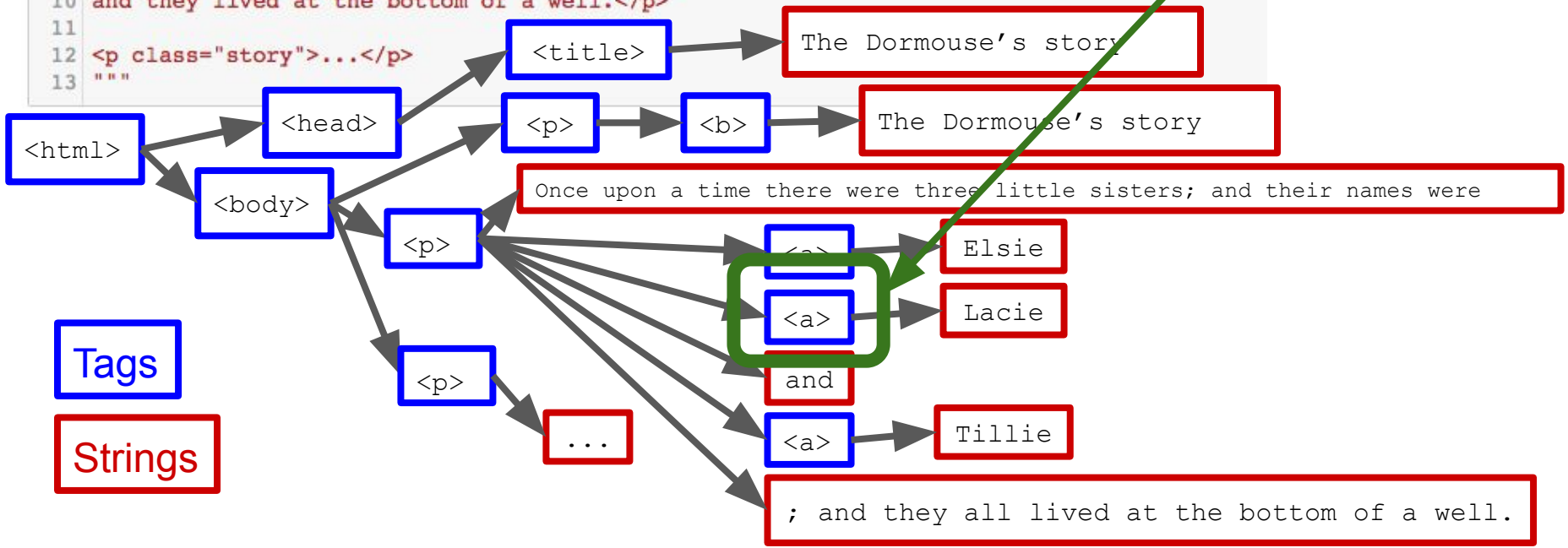
```
1  html_doc = """
2  <html><head><title>The Dormouse's story</title></head>
3  <body>
4  <p class="title"><b>The Dormouse's story</b></p>
5
6  <p class="story">Once upon a time there were three little sisters; and their names were
7  <a href="http://example.com/elsie" class="sister" id="link1">Elsie</a>,
8  <a href="http://example.com/lacie" class="sister" id="link2">Lacie</a> and
9  <a href="http://example.com/tillie" class="sister" id="link3">Tillie</a>;
10 and they lived at the bottom of a well.</p>
11
12 <p class="story">...</p>
13 """
```



HTML tree structure

```
1  html_doc = """
2  <html><head><title>The Dormouse's story</title></head>
3  <body>
4  <p class="title"><b>The Dormouse's story</b></p>
5
6  <p class="story">Once upon a time there were three little sisters; and their names were
7  <a href="http://example.com/elsie" class="sister" id="link1">Elsie</a>,
8  <a href="http://example.com/lacie" class="sister" id="link2">Lacie</a> and
9  <a href="http://example.com/tillie" class="sister" id="link3">Tillie</a>;
10 and they lived at the bottom of a well.</p>
11
12 <p class="story">...</p>
13 """
```

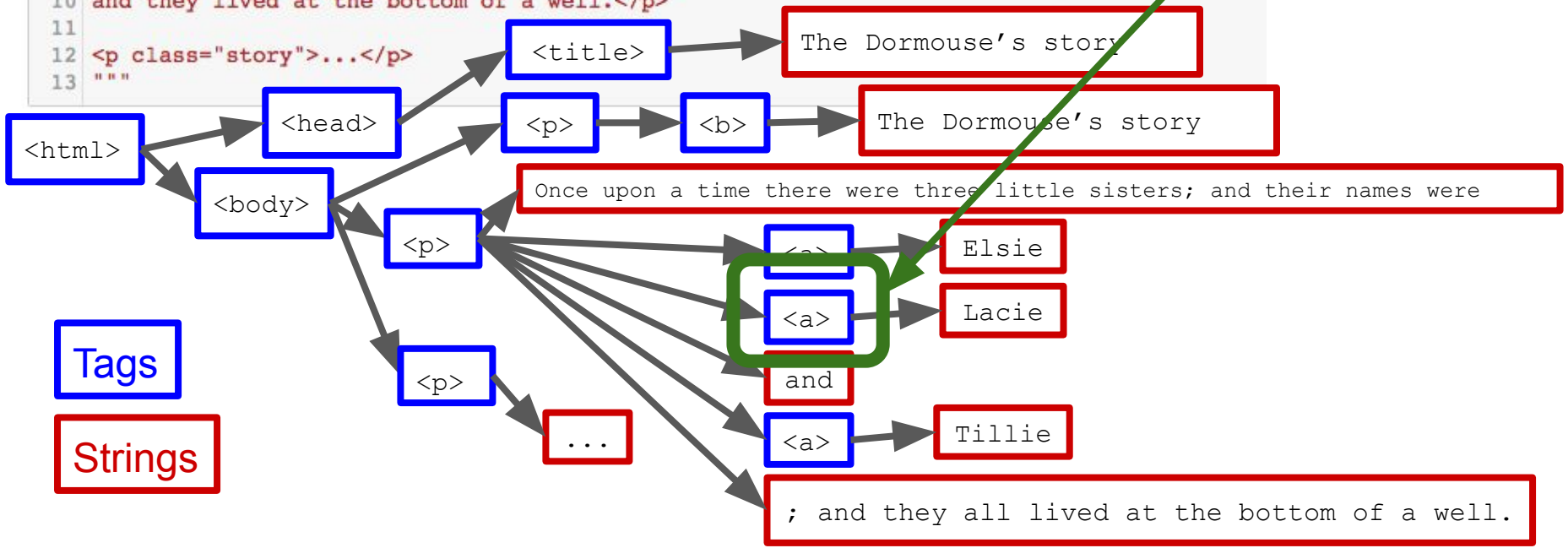
Question: what are the attributes of this node in the tree? That is, what are the attributes of this tag?



HTML tree structure

```
html_doc = """
<html><head><title>The Dormouse's story</title></head>
<body>
<p class="title"><b>The Dormouse's story</b></p>

<p class="story">Once upon a time there were three little sisters; and their names were
<a href="http://example.com/elsie" class="sister" id="link1">Elsie</a>,
<a href="http://example.com/lacie" class="sister" id="link2">Lacie</a> and
<a href="http://example.com/tillie" class="sister" id="link3">Tillie</a>;
and they lived at the bottom of a well.</p>

<p class="story">...</p>
"""
```

Question: what are the attributes of this node in the tree? That is, what are the attributes of this tag?



Navigating the HTML tree

```
1 parsed.title
```

```
<title>The Dormouse's story</title>
```

```
1 parsed.title.string
```

```
u"The Dormouse's story"
```

```
1 parsed.a
```

```
<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>
```

```
1 parsed.p
```

```
<p class="title"><b>The Dormouse's story</b></p>
```

```
1 parsed.p.b
```

```
<b>The Dormouse's story</b>
```

Tag name gets the first tag of that type in the tree.

If a tag's child is a string, access it with `tag.string`

Can go down the tree by asking for tags of tags of...

Navigating the HTML tree

```
shortdoc="""
<p class="story">Once upon a time there were three little sisters; and their names were
<a href="http://example.com/elsie" class="sister" id="link1">Elsie</a>,
<a href="http://example.com/lacie" class="sister" id="link2">Lacie</a> and
<a href="http://example.com/tillie" class="sister" id="link3">Tillie</a>;
and they lived at the bottom of a well.</p>
"""
pshort = BeautifulSoup(shortdoc, 'html.parser')
pshort.p.contents
```

Access a list of children of a tag with `.contents`

```
[u'Once upon a time there were three little sisters; and their names were\n',
 <a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>,
 u',\n',
 <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>,
 u' and\n',
 <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>,
 u';\nand they lived at the bottom of a well.']
```

```
pshort.p.children
```

Or get the same information in a Python iterator with `.children`

```
<listiterator at 0x1129d2690>
```

```
pshort.p.descendants
```

Recurse down the whole tree with `.descendants`

```
<generator object descendants at 0x1129bd410>
```

Navigating the HTML tree

The tree structure means that every tag has a parent (except the “root” tag, which has parent “None”).

```
1 link = parsed.a
2 link
```

```
<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>
```

Access a tag’s parent tag with `.parent`

```
1 link.parent
```

```
<p class="story">Once upon a time there were three little sisters; and their names were\n<a class="sister" href="http
://example.com/elsie" id="link1">Elsie</a>,\n<a class="sister" href="http://example.com/lacie" id="link2">Lacie</a> a
nd\n<a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>;\nand they lived at the bottom of a well
.</p>
```

Get the whole chain of parents back to the root with `.parents`

```
1 for parent in link.parents:
2     print(parent.name)
```

```
p
body
html
[document]
```

Move “left and right” in the tree with `.previous_sibling` and `.next_sibling`

```
1 link.previous_sibling
```

```
u'Once upon a time there were three little sisters; and their names were\n'
```

```
1 link.next_sibling
```

```
u',\n'
```

Searching the tree: `find_all` and related methods

```
1 parsed = BeautifulSoup(html_doc, 'html.parser')
2
3 parsed.find_all('p')
```

Finds all tags with name `'p'`

```
[<p class="title"><b>The Dormouse's story</b></p>,
 <p class="story">Once upon a time there were three little sisters; and their names were\n<a class="sister" href="htt
p://example.com/elsie" id="link1">Elsie</a>,\n<a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>
and\n<a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>;\nand they lived at the bottom of a wel
l.</p>,
 <p class="story">...</p>]
```

```
3 parsed.find_all(['a','b'])
```

Finds all tags with names matching **either** `'a'` or `'b'`

```
[<b>The Dormouse's story</b>,
 <a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>,
 <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>,
 <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>]
```

```
4 import re
5 parsed.find_all(re.compile(r'^b'))
```

Finds all tags whose names match the given regex.

```
[<body>\n<p class="title"><b>The Dormouse's story</b></p>\n<p class="story">Once upon a time there were three little
sisters; and their names were\n<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>,\n<a class="sis
ter" href="http://example.com/lacie" id="link2">Lacie</a> and\n<a class="sister" href="http://example.com/tillie" id=
"link3">Tillie</a>;\nand they lived at the bottom of a well.</p>\n<p class="story">...</p>\n</body>,
 <b>The Dormouse's story</b>]
```

More about `find_all`

```
 8 def has_class_but_no_id(tag):
 9     return tag.has_attr('class') and not tag.has_attr('id')
10
11 parsed.find_all(has_class_but_no_id)
```

```
[<p class="title"><b>The Dormouse's story</b></p>,
 <p class="story">Once upon a time there were three little sisters; and their names were\n<a class="sister" href="htt
p://example.com/elsie" id="link1">Elsie</a>,\n<a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>
and\n<a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>;\nand they lived at the bottom of a wel
l.</p>,
 <p class="story">...</p>]
```

Pass in a function that returns `True`/`False` given a tag, and `find_all` will return only the tags that evaluate `True`

Note: by default, `find_all` recurses down the whole tree, but you can have it only search the immediate children of the tag by passing the flag `recursive=False`.

See https://www.crummy.com/software/BeautifulSoup/bs4/doc/#find-all for more.

Flattening contents: `get_text()`

```
1 shortdoc="""
2 <p class="story">Once upon a time there were three little sisters; and their names were
3 <a href="http://example.com/elsie" class="sister" id="link1">Elsie</a>,
4 <a href="http://example.com/lacie" class="sister" id="link2">Lacie</a> and
5 <a href="http://example.com/tillie" class="sister" id="link3">Tillie</a>;
6 and they lived at the bottom of a well.</p>
7 """
8 pshort = BeautifulSoup(shortdoc, 'html.parser')
9 print pshort.p.string is None
```

```
True
```

This `<p>` tag contains a full sentence, but some parts of that sentence are links, so `p.string` fails. What do I do if I want to get the full string without the links?

```
1 pshort.p.contents
```

```
[u'Once upon a time there were three little sisters; and their names were\n',
 <a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>,
 u',\n',
 <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>,
 u' and\n',
 <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>,
 u';\nand they lived at the bottom of a well.']
```

Note: common cause of bugs/errors in `BeautifulSoup` is trying to access `tag.string` when it doesn't exist!

```
1 pshort.p.get_text()
```

```
u'Once upon a time there were three little sisters; and their names were\nElsie,\nLacie and\nTillie;\nand they lived
at the bottom of a well.'
```

XML - eXtensible Markup Language, .xml

https://en.wikipedia.org/wiki/XML

Core idea: separate data from its presentation

Note that HTML *doesn't* do this-- the HTML for the webpage **is** the data

But XML is tag-based, very similar to HTML

BeautifulSoup will parse XML

https://www.crummy.com/software/BeautifulSoup/bs4/doc/#installing-a-parser

We won't talk much about XML, because it's falling out of favor, replaced by...

JSON - JavaScript Object Notation

https://en.wikipedia.org/wiki/JSON

Commonly used by website APIs

Basic building blocks:

- attribute–value pairs
- array data

Example (right) from wikipedia:

- Possible JSON representation of a person

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    },
    {
      "type": "mobile",
      "number": "123 456-7890"
    }
  ],
  "children": [],
  "spouse": null
}
```

Python `json` module

```
1 import json
2 json_string = '{"first_name":"John",  "last_name":"Bardeen", "alma_mater":"University of Wisconsin"}'
3 parsed_json = json.loads(json_string)
4 parsed_json
```

```
{'alma_mater': 'University of Wisconsin',
 'first_name': 'John',
 'last_name': 'Bardeen'}
```

```
1 json.dumps(parsed_json)
```

```
'{"first_name": "John", "last_name": "Bardeen", "alma_mater": "University of Wisconsin"}'
```

JSON string encoding information about physicist John Bardeen

`json.loads` parses a string and returns a JSON object.

`json.dumps` turns a JSON object back into a string.

Python `json` module

```
1 parsed_json
```

```
{'alma_mater': 'University of Wisconsin',
 'first_name': 'John',
 'last_name': 'Bardeen'}
```

```
1 parsed_json['alma_mater']
```

```
'University of Wisconsin'
```

```
1 parsed_json['first_name']
```

```
'John'
```

```
1 parsed_json['middle_name']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-9-e0447f76c1d5> in <module>()
----> 1 parsed_json['middle_name']

KeyError: 'middle_name'
```

JSON object returned by `json.loads` acts just like a Python dictionary.

JSON objects can have very complicated structure

```
complex_json_string="""{
    "id": "0001",
    "type": "donut",
    "name": "Cake",
    "ppu": 0.55,
    "batters":
        {
            "batter":
                [
                    { "id": "1001", "type": "Regular" },
                    { "id": "1002", "type": "Chocolate" },
                    { "id": "1003", "type": "Blueberry" },
                    { "id": "1004", "type": "Devil's Food" }
                ]
        },
    "topping":
        [
            { "id": "5001", "type": "None" },
            { "id": "5002", "type": "Glazed" },
            { "id": "5005", "type": "Sugar" },
            { "id": "5007", "type": "Powdered Sugar" },
            { "id": "5006", "type": "Chocolate with Sprinkles" },
            { "id": "5003", "type": "Chocolate" },
            { "id": "5004", "type": "Maple" }
        ]
}"""
```

JSON objects can have very complicated structure

```
complex_json_string="""{
    "id": "0001",
    "type": "donut",
    "name": "Cake",
    "ppu": 0.55,
    "batters":
        {
            "batter":
                [
                    { "id": "1001", "type": "Regular" },
                    { "id": "1002", "type": "Chocolate" },
                    { "id": "1003", "type": "Blueberry" },
                    { "id": "1004", "type": "Devil's Food" }
                ]
        },
    "topping":
        [
            { "id": "5001", "type": "None" },
            { "id": "5002", "type": "Glazed" },
            { "id": "5005", "type": "Sugar" },
            { "id": "5007", "type": "Powdered Sugar" },
            { "id": "5006", "type": "Chocolate with Sprinkles" },
            { "id": "5003", "type": "Chocolate" },
            { "id": "5004", "type": "Maple" }
        ]
}"""
```

This can get out of hand quickly, if you're trying to work with large collections of data. For an application like that, you are better off using a database, about which we'll learn in our next lecture.