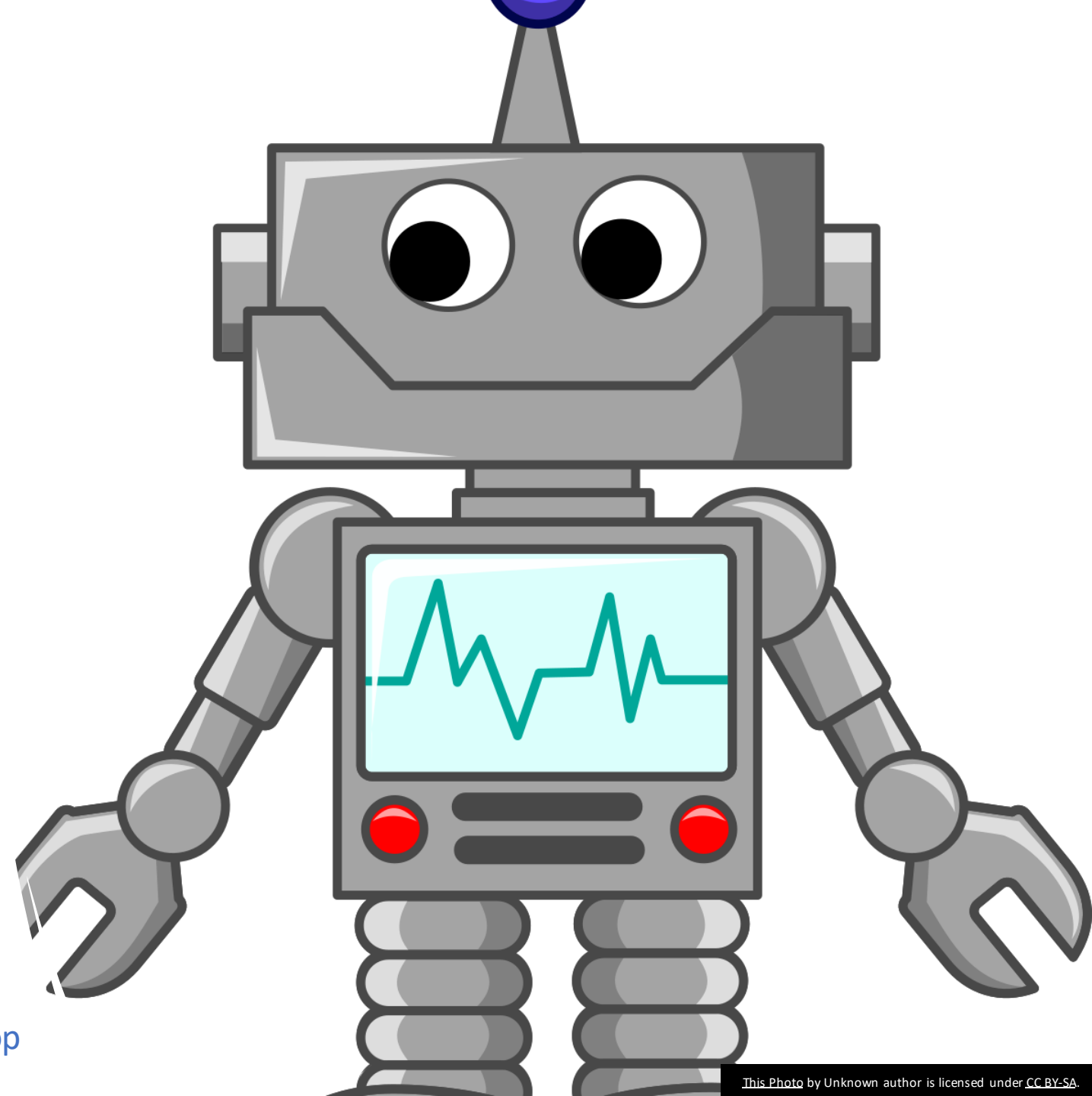
Python Workshop III:
Web Scraping

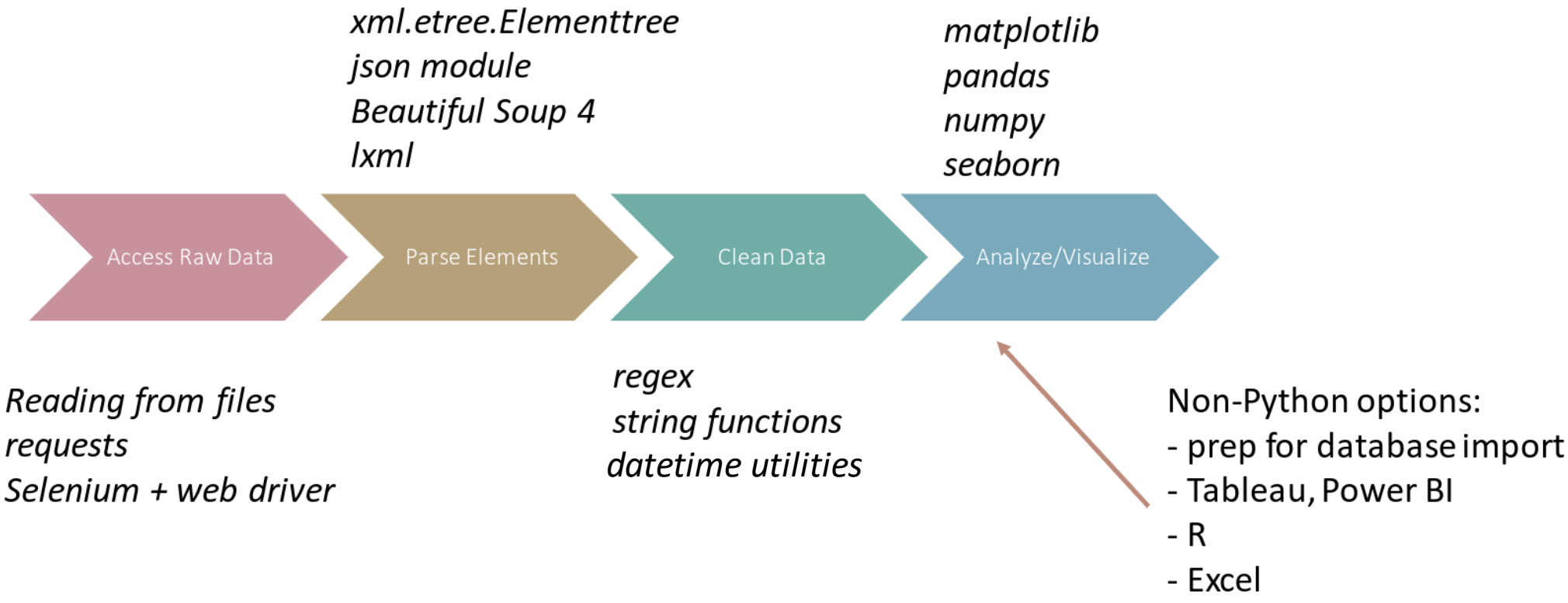
Monica Ihli

monica@utk.edu

https://github.com/monicaihli/python-data-workshop



Data Processing Workflow



1. Get the raw data on your computer.
2. Extract into Python data structures
3. Use Python tools to clean up and restructure usable format.
4. Analyze the data

Python requests Library

- Third-party library, must install.
- Convenient third-party library which simplifies handling most aspects of working with **HTTP**.

Some HTTP Basics

HTTP Methods indicate the desired action. Examples:

GET – retrieve the requested resource.

POST - Submit data to the resource.

PUT – Update the target resource with the current payload.

"Payload" refers to the data sent over an HTTP request.

HTTP Headers

GENERAL headers like *Date* or *Connection* apply to request or response, but not to the content of what is being sent.

REQUEST headers like Accept, User-Agent

RESPONSE headers like *Age* or *Server* give additional context about the response.

ENTITY headers like *Content-Length, Content-Encoding* describe content of the message body.

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

Common Status Codes

200 – Ok *

404 – Not Found

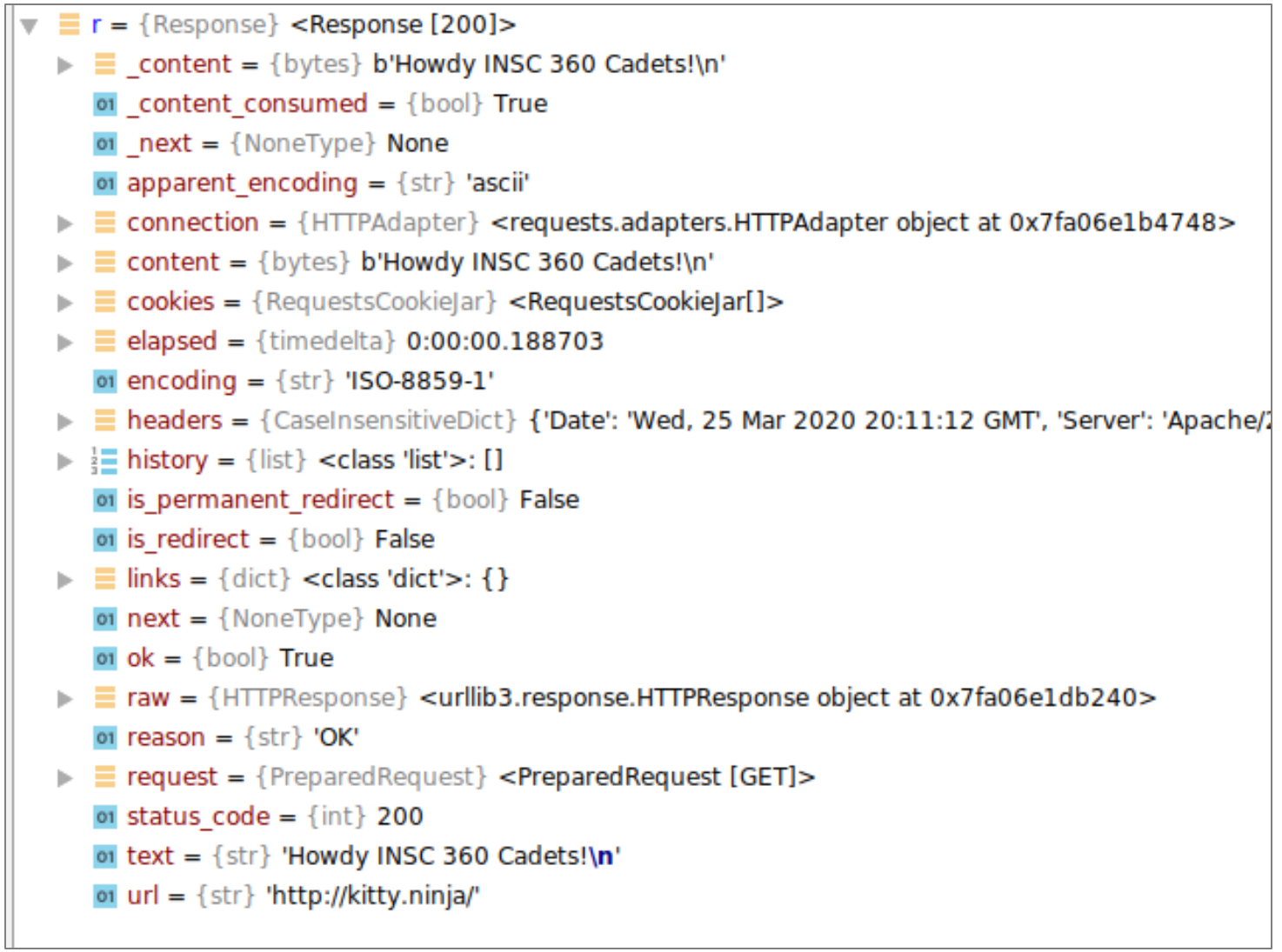
503 – Service unavailable

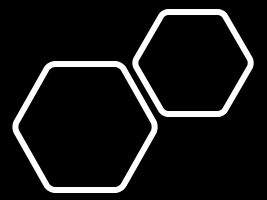
500 – Internal Server error

* means no error in http request but problems can happen on other levels!

A simple request

```
import requests
r = requests.get("http://kitty.ninja")
```





Request Arguments

url- You can provide a domain-based or even an IP address:

```
url='http://167.99.155.69')
```

params – a dict of key-value pairs representing HTTP queries and inputs:

```
payload = {'verb': 'ListRecords', 'from': '2020-03-25' }
```

headers – constructed using a dict of key-value pairs:

```
headers = {'user-agent': 'Mozilla/5.0 (X11; Ubuntu; Linux
x86_64; rv:74.0) Gecko/20100101 Firefox/74.0'} # haha look
like a human!
```

A More Complicated Request

A dictionary of key-value pairs stored in a variable called "payload", which contains search parameters.

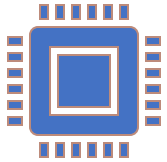
```
r = requests.get(url=baseurl, params=payload, headers=headers)
```

Another dictionary of key-value pairs containing ordinary HTTP headers.

Web address to pull content from

Web Scraping

- Web scraping usually refers to extracting data from web pages, often in a fashion that simulates human behavior.
- Web scraping often entails heavy reliance on string-processing functions to clean data extracted from HTML, which is far more unruly and wild than the neatly structured machine-readable content exposed in APIs.



Web Scraping: The Complete Package

- **Selenium** – Python package implements tools for controlling a web driver.
- **Chromedriver** (or other web driver) - tool for automated testing of webapps. It supports navigating to web pages, user input, executes JavaScript, etc.
- **Google Chrome** – It is possible with other browsers but this is what we will use.
- **Beautiful Soup** – a Python library for pulling data out of HTML and *XML files.

** Note: If you want to use for XML parsing, will need to install lxml package*

Web Scraping w/ Selenium

- Sometimes web pages aren't fully rendered in the moment you make the request. A page with JavaScript can take time to finish processing and loading the content.
- This means that whatever you get back from that HTTP request isn't going to have the full page content.
- Solution: Use Selenium & chromedriver-- Take control of a browser (Chrome) programmatically using a special driver.
- Allows page to fully load before retrieving its contents.

Capturing Raw Data Over HTTP

REQUESTS

- Faster capture over HTTP.
- Simpler to implement.
- Can't interact with page.
- Won't run javascript.
- No default user-agent.
- Best choice for web services.

SELENIUM/WEB DRIVER

- Can interact with webpage to get to desired data such by navigating and clicking buttons.
- Will execute JavaScript.
- Slower than requests because you are operating through a browser.
- Browser's default user-agent
- Best choice for web pages with JS or needing interaction.

Create Web Driver & Fetch Page Source

```
url = "http://kitty.ninja/demo.html"
driver = webdriver.Chrome(executable_path="~/Downloads/webdrivers/chromedriver")
driver.get(url)
```

Store Document Tree in Memory

```
soup = BeautifulSoup(driver.page_source, "html.parser")
```

About Beautiful Soup 4

- To parse a document, pass either a string or a file handler to the BeautifulSoup constructor
- Content is first converted to Unicode.
- Transforms a complex HTML document into a complex tree of Python objects.
- The 4 objects you will interact with are:
  - **Tag** – Corresponds to an html or xml element; Has name and attributes
  - **NavigableString** – Contains text within a tag
  - **BeautifulSoup** – represents the parsed document as a whole
  - **Comment** – special type of NavigableString

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

bs4 parser options

This table summarizes the advantages and disadvantages of each parser library:

Parser	Typical usage	Advantages	Disadvantages
Python's html.parser	`BeautifulSoup(markup, "html.parser")`	• Batteries included • Decent speed • Lenient (As of Python 2.7.3 and 3.2.)	• Not as fast as lxml, less lenient than html5lib.
lxml's HTML parser	`BeautifulSoup(markup, "lxml")`	• Very fast • Lenient	• External C dependency
lxml's XML parser	`BeautifulSoup(markup, "lxml-xml")` `BeautifulSoup(markup, "xml")`	• Very fast • The only currently supported XML parser	• External C dependency
html5lib	`BeautifulSoup(markup, "html5lib")`	• Extremely lenient • Parses pages the same way a web browser does • Creates valid HTML5	• Very slow • External Python dependency

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

BS4: Navigating the Tree

- Tags may contain strings and other tags as children.
- Can navigate using tag names (such as `soup.p` or `soup.title`)
- Can access tag attributes similarly to a dictionary. (`tag['id']`)
- **`soup.find_all('p')`** <- Would return a set of tags which match p (paragraph)
- All a tags children are available in a list as a property of the tag called **.contents**
- Can also access children of a tag through a generator:

```
for child in title_tag.children:
```

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

Parse Elements

```
for child in soup.children:
    print(child)
```

```
<!DOCTYPE html>
<html lang="en">
<head>…  </head>
<body>
   <h2>Top</h2>
   <p class="top">This is the first paragraph</p>
   <p class="top">This is the second
paragraph</p>
   <h2>Middle</h2>
   <p>This is the <b>middle</b> paragraph</p>
   <h2>Bottom</h2>
   <p class="bottom">This is the fourth
paragraph</p>
   <p class="bottom">This is the fifth
paragraph</p>
</body>
</html>
```

Parse Elements

```
paragraphs = soup.find_all('p')
for p in paragraphs:
    print(p.text)
    if p.b:
        print("{} is in BOLD!".format(p.b))
```

```
<!DOCTYPE html>
<html lang="en">
<head>…  </head>
<body>
   <h2>Top</h2>
   <p class="top">This is the first paragraph</p>
   <p class="top">This is the second
paragraph</p>
   <h2>Middle</h2>
   <p>This is the <b>middle</b> paragraph</p>
   <h2>Bottom</h2>
   <p class="bottom">This is the fourth
paragraph</p>
   <p class="bottom">This is the fifth
paragraph</p>
</body>
</html>
```