# Scrape the Web: Strategies for programming websites that don't expect it

Presenter: Asheesh Laroia (scrape-pycon@asheesh.org, +1-585-506-8865)

March 26, 2009

# Outline

# Meta

# Introduction

(screen is in presentation-only mode)

# Introduction

(screen is in presentation-only mode)

- You will learn neat tricks

# Introduction

(screen is in presentation-only mode)

- You will learn neat tricks
- DO NOT BECOME AN EVIL COMMENT SPAMMER

# Introduction

(screen is in presentation-only mode)

- You will learn neat tricks
- DO NOT BECOME AN EVIL COMMENT SPAMMER
- Theory, practice, and iterative development

# Introduction

(screen is in presentation-only mode)

- You will learn neat tricks
- DO NOT BECOME AN EVIL COMMENT SPAMMER
- Theory, practice, and iterative development
- Brittle? Sometimes.

# Introduction

(screen is in presentation-only mode)

- You will learn neat tricks
- DO NOT BECOME AN EVIL COMMENT SPAMMER
- Theory, practice, and iterative development
- Brittle? Sometimes.
- The comics aren't mine; ask me for references.

# Format introduction (5 minutes)

(screen switches from presentation-only mode to presentation+agenda+chat)

# Format introduction (5 minutes)

(screen switches from presentation-only mode to presentation+agenda+chat)

- Format: Interactive agenda and “backchannel” chat always visible

# Format introduction (5 minutes)

(screen switches from presentation-only mode to presentation+agenda+chat)

- Format: Interactive agenda and “backchannel” chat always visible
- Go to http://pycon09.asheesh.org/ and:

# Format introduction (5 minutes)

(screen switches from presentation-only mode to presentation+agenda+chat)

- Format: Interactive agenda and "backchannel" chat always visible
- Go to http://pycon09.asheesh.org/ and:
  - join the IRC chat (keep that URL visible in the chat window; make it the topic)

# Format introduction (5 minutes)

(screen switches from presentation-only mode to presentation+agenda+chat)

- Format: Interactive agenda and "backchannel" chat always visible
- Go to http://pycon09.asheesh.org/ and:
  - join the IRC chat (keep that URL visible in the chat window; make it the topic)
  - install FireBug

# Format introduction (5 minutes)

(screen switches from presentation-only mode to presentation+agenda+chat)

- Format: Interactive agenda and "backchannel" chat always visible
- Go to http://pycon09.asheesh.org/ and:
  - join the IRC chat (keep that URL visible in the chat window; make it the topic)
  - install FireBug
- We will not proceed until many/most people have joined

# “Only” three hours (2 minutes)

# “Only” three hours (2 minutes)

- Slow me down,

# “Only” three hours (2 minutes)

- Slow me down,
- or speed me up.

# “Only” three hours (2 minutes)

- Slow me down,
- or speed me up.
- Do this with your voice, by raising your hand, or in chat.

# What is web scraping? (2 minutes)

Generally speaking,

# What is web scraping? (2 minutes)

Generally speaking,

1. You retrieve some page from the web,

# What is web scraping? (2 minutes)

Generally speaking,

1. You retrieve some page from the web,
2. You extract some information,

# What is web scraping? (2 minutes)

Generally speaking,

1. You retrieve some page from the web,
2. You extract some information,
3. and optionally you repeat.

# When web scrape? (2 minutes)

# When web scrape? (2 minutes)

- If you need data right now (like curry), or

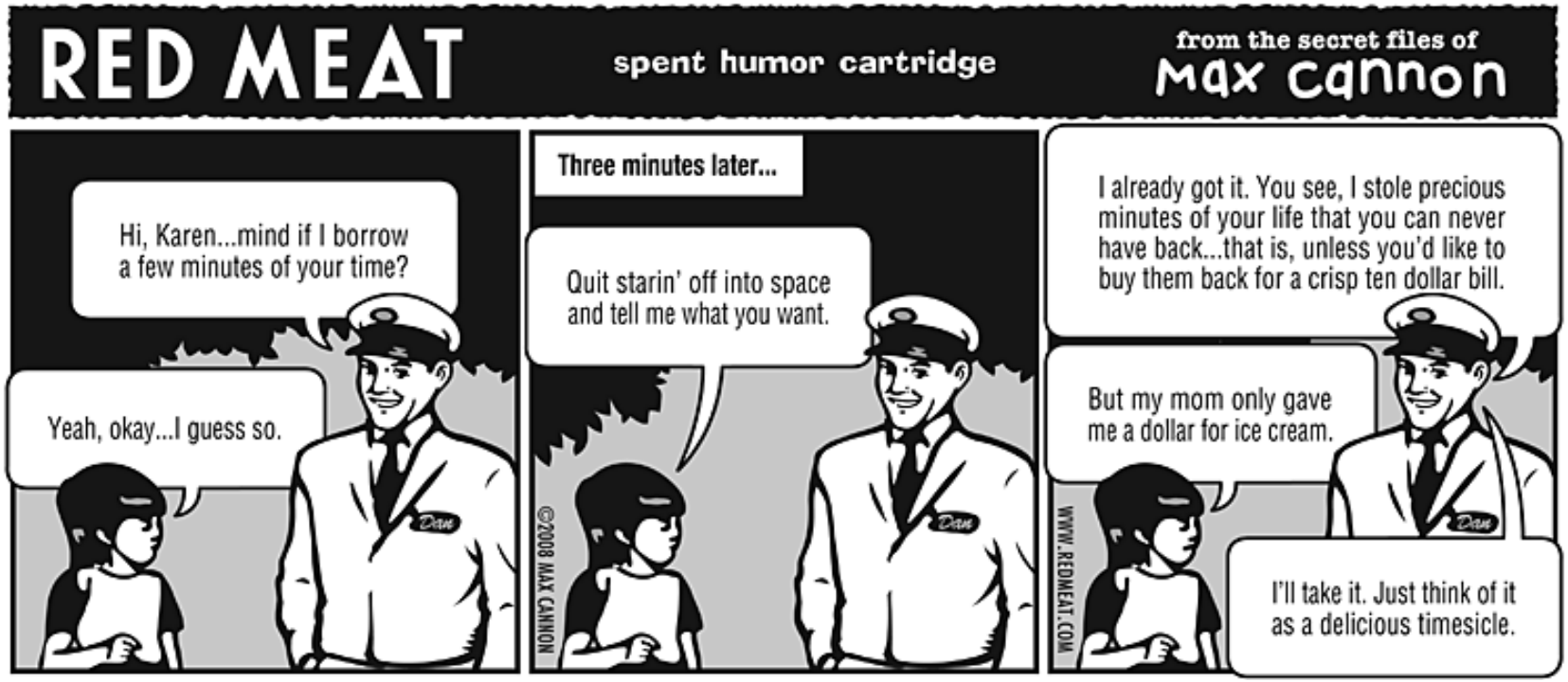
# When web scrape? (2 minutes)

- If you need data right now (like curry), or
- if you will need the data later.

# When web scrape? (2 minutes)

- If you need data right now (like curry), or
- if you will need the data later.
- See Cathy comic:
  http://www.gocomics.com/cathy/2009/02/11/

# The point of this presentation



# Outline

# Example 1

# Download and analyze curry (5 minutes)

First, define the problem

# Download and analyze curry (5 minutes)

First, define the problem

- Load up http://mehfilindian.com/LunchMenuTakeOut.htm

# Download and analyze curry (5 minutes)

First, define the problem

- Load up http://mehfilindian.com/LunchMenuTakeOut.htm
- Decide the question we want to answer

# Download and analyze curry (5 minutes)

First, define the problem

- Load up http://mehfilindian.com/LunchMenuTakeOut.htm
- Decide the question we want to answer
  - "Is there eggplant?"

# Now, from Python

examples/curry/trivial.py

# Now, from Python

examples/curry/trivial.py

- urllib2.urlopen() gives you a file descriptor

# Now, from Python

examples/curry/trivial.py

- urllib2.urlopen() gives you a file descriptor
- Now you can read() it... (and you get a big ol' string)

# Now, from Python

examples/curry/trivial.py

- urllib2.urlopen() gives you a file descriptor
- Now you can read() it... (and you get a big ol' string)
- test its contents for eggplant

# “Extract some information” (8 minutes)

# “Extract some information” (8 minutes)

- HTML

# “Extract some information” (8 minutes)

- HTML
- vs. XHTML (2000)

# “Extract some information” (8 minutes)

- HTML
- vs. XHTML (2000)
- Both are trees of tags; both can be visualized in FireBug.

# “Extract some information” (8 minutes)

- HTML
- vs. XHTML (2000)
- Both are trees of tags; both can be visualized in FireBug.
- ...did XHTML win?

# Outline

# Stats pop quiz

(Stats from the MAMA survey published by Opera <http://dev.opera.com/articles/view/mama-key-findings/>.)

# Stats pop quiz

(Stats from the MAMA survey published by Opera <http://dev.opera.com/articles/view/mama-key-findings/>.)

- Average page size?

# Stats pop quiz

(Stats from the MAMA survey published by Opera <http://dev.opera.com/articles/view/mama-key-findings/>.)

- Average page size?
  - 16.5K

# Stats pop quiz

(Stats from the MAMA survey published by Opera <http://dev.opera.com/articles/view/mama-key-findings/>.)

- Average page size?
  - 16.5K
- HTML to XHTML ratio?

# Stats pop quiz

(Stats from the MAMA survey published by Opera <http://dev.opera.com/articles/view/mama-key-findings/>.)

- Average page size?
  - 16.5K
- HTML to XHTML ratio?
  - 2:1

# Stats pop quiz

(Stats from the MAMA survey published by Opera <http://dev.opera.com/articles/view/mama-key-findings/>.)

- Average page size?
  - 16.5K
- HTML to XHTML ratio?
  - 2:1
- Transitional vs. Strict/Frameset:

# Stats pop quiz

(Stats from the MAMA survey published by Opera <http://dev.opera.com/articles/view/mama-key-findings/>.)

- Average page size?
  - 16.5K
- HTML to XHTML ratio?
  - 2:1
- Transitional vs. Strict/Frameset:
  - 10:1

# Stats pop quiz

(Stats from the MAMA survey published by Opera <http://dev.opera.com/articles/view/mama-key-findings/>.)

- Average page size?
  - 16.5K
- HTML to XHTML ratio?
  - 2:1
- Transitional vs. Strict/Frameset:
  - 10:1
- How many in "Quirks" mode?

# Stats pop quiz

(Stats from the MAMA survey published by Opera <http://dev.opera.com/articles/view/mama-key-findings/>.)

- Average page size?
  - 16.5K
- HTML to XHTML ratio?
  - 2:1
- Transitional vs. Strict/Frameset:
  - 10:1
- How many in "Quirks" mode?
  - 85%

# Stats pop quiz

(Stats from the MAMA survey published by Opera <http://dev.opera.com/articles/view/mama-key-findings/>.)

- Average page size?
  - 16.5K
- HTML to XHTML ratio?
  - 2:1
- Transitional vs. Strict/Frameset:
  - 10:1
- How many in "Quirks" mode?
  - 85%
- What's more popular? TITLE or BODY?

# Stats pop quiz

(Stats from the MAMA survey published by Opera <http://dev.opera.com/articles/view/mama-key-findings/>.)

- Average page size?
  - 16.5K
- HTML to XHTML ratio?
  - 2:1
- Transitional vs. Strict/Frameset:
  - 10:1
- How many in "Quirks" mode?
  - 85%
- What's more popular? TITLE or BODY?
  - TITLE

# Stats pop quiz

(Stats from the MAMA survey published by Opera <http://dev.opera.com/articles/view/mama-key-findings/>.)

- Average page size?
  - 16.5K
- HTML to XHTML ratio?
  - 2:1
- Transitional vs. Strict/Frameset:
  - 10:1
- How many in "Quirks" mode?
  - 85%
- What's more popular? TITLE or BODY?
  - TITLE
- What percent validate in general?

# Stats pop quiz

(Stats from the MAMA survey published by Opera <http://dev.opera.com/articles/view/mama-key-findings/>.)

- Average page size?
  - 16.5K
- HTML to XHTML ratio?
  - 2:1
- Transitional vs. Strict/Frameset:
  - 10:1
- How many in "Quirks" mode?
  - 85%
- What's more popular? TITLE or BODY?
  - TITLE
- What percent validate in general?
  - ca. 4.13%

# Stats pop quiz

(Stats from the MAMA survey published by Opera <http://dev.opera.com/articles/view/mama-key-findings/>.)

- Average page size?
  - 16.5K
- HTML to XHTML ratio?
  - 2:1
- Transitional vs. Strict/Frameset:
  - 10:1
- How many in "Quirks" mode?
  - 85%
- What's more popular? TITLE or BODY?
  - TITLE
- What percent validate in general?
  - ca. 4.13%
- What percent of web pages that have validation badges validate?

# Stats pop quiz

(Stats from the MAMA survey published by Opera <http://dev.opera.com/articles/view/mama-key-findings/>.)

- Average page size?
  - 16.5K
- HTML to XHTML ratio?
  - 2:1
- Transitional vs. Strict/Frameset:
  - 10:1
- How many in "Quirks" mode?
  - 85%
- What's more popular? TITLE or BODY?
  - TITLE
- What percent validate in general?
  - ca. 4.13%
- What percent of web pages that have validation badges validate?
  - ca. $\frac{1}{2}$

# Outline

# Parsing considerations

# A showcase of some of your options

# A showcase of some of your options

- An example of valid HTML (written by hand) (examples/parsing/)

# A showcase of some of your options

- An example of valid HTML (written by hand) (examples/parsing/)
  - Parsed with HTMLParser

# A showcase of some of your options

- An example of valid HTML (written by hand) (examples/parsing/)
  - Parsed with HTMLParser
- An example of invalid HTML (cooked by hand) (examples/parsing/invalid-html/)

# A showcase of some of your options

- An example of valid HTML (written by hand) (examples/parsing/)
  - Parsed with HTMLParser
- An example of invalid HTML (cooked by hand) (examples/parsing/invalid-html/)
  - Parsed with HTMLParser

# A showcase of some of your options

- An example of valid HTML (written by hand) (examples/parsing/)
  - Parsed with HTMLParser
- An example of invalid HTML (cooked by hand) (examples/parsing/invalid-html/)
  - Parsed with HTMLParser
- An example of valid XHTML (written by hand) (examples/parsing/valid-xhtml/)

# A showcase of some of your options

- An example of valid HTML (written by hand) (examples/parsing/)
  - Parsed with HTMLParser
- An example of invalid HTML (cooked by hand) (examples/parsing/invalid-html/)
  - Parsed with HTMLParser
- An example of valid XHTML (written by hand) (examples/parsing/valid-xhtml/)
  - Parsed with xml.dom.minidom

# A showcase of some of your options

- An example of valid HTML (written by hand) (examples/parsing/)
  - Parsed with HTMLParser
- An example of invalid HTML (cooked by hand) (examples/parsing/invalid-html/)
  - Parsed with HTMLParser
- An example of valid XHTML (written by hand) (examples/parsing/valid-xhtml/)
  - Parsed with xml.dom.minidom
  - Parsed with HTMLParser

# A showcase of some of your options

- An example of valid HTML (written by hand) (examples/parsing/)
  - Parsed with HTMLParser
- An example of invalid HTML (cooked by hand) (examples/parsing/invalid-html/)
  - Parsed with HTMLParser
- An example of valid XHTML (written by hand) (examples/parsing/valid-xhtml/)
  - Parsed with xml.dom.minidom
  - Parsed with HTMLParser
- An example of invalid XHTML <http://www.washington.edu/accessit/webdesign/student/unit5/inv (examples/parsing/invalid-xhtml/)

# A showcase of some of your options

- An example of valid HTML (written by hand) (examples/parsing/)
  - Parsed with HTMLParser
- An example of invalid HTML (cooked by hand) (examples/parsing/invalid-html/)
  - Parsed with HTMLParser
- An example of valid XHTML (written by hand) (examples/parsing/valid-xhtml/)
  - Parsed with xml.dom.minidom
  - Parsed with HTMLParser
- An example of invalid XHTML <http://www.washington.edu/accessit/webdesign/student/unit5/inv (examples/parsing/invalid-xhtml/)
  - in Firefox

# A showcase of some of your options

- An example of valid HTML (written by hand) (examples/parsing/)
  - Parsed with HTMLParser
- An example of invalid HTML (cooked by hand) (examples/parsing/invalid-html/)
  - Parsed with HTMLParser
- An example of valid XHTML (written by hand) (examples/parsing/valid-xhtml/)
  - Parsed with xml.dom.minidom
  - Parsed with HTMLParser
- An example of invalid XHTML <http://www.washington.edu/accessit/webdesign/student/unit5/inv (examples/parsing/invalid-xhtml/)
  - in Firefox
  - In xml.dom.minidom

# A showcase of some of your options

- An example of valid HTML (written by hand) (examples/parsing/)
  - Parsed with HTMLParser
- An example of invalid HTML (cooked by hand) (examples/parsing/invalid-html/)
  - Parsed with HTMLParser
- An example of valid XHTML (written by hand) (examples/parsing/valid-xhtml/)
  - Parsed with xml.dom.minidom
  - Parsed with HTMLParser
- An example of invalid XHTML <http://www.washington.edu/accessit/webdesign/student/unit5/inv (examples/parsing/invalid-xhtml/)
  - in Firefox
  - In xml.dom.minidom
  - in HTMLParser

# A showcase of some of your options

- An example of valid HTML (written by hand) (examples/parsing/)
  - Parsed with HTMLParser
- An example of invalid HTML (cooked by hand) (examples/parsing/invalid-html/)
  - Parsed with HTMLParser
- An example of valid XHTML (written by hand) (examples/parsing/valid-xhtml/)
  - Parsed with xml.dom.minidom
  - Parsed with HTMLParser
- An example of invalid XHTML <http://www.washington.edu/accessit/webdesign/student/unit5/inv (examples/parsing/invalid-xhtml/)
  - in Firefox
  - In xml.dom.minidom
  - in HTMLParser
- If web HTML is not always parseable, we need a different approach

# Other ways to get information out of web pages? (1 minute)

# Other ways to get information out of web pages? (1 minute)

- “eggplant” in page_contents.lower()

# Other ways to get information out of web pages? (1 minute)

- “eggplant” in page_contents.lower()
- re.search(“eggplant”, page_contents, re.IGNORECASE)

# Inspirational quote: JWZ (1 minute)

*Some people, when confronted with a problem, think "I know, I'll use regular expressions." Now they have two problems.*
*– Jamie Zawinski*

# What's wrong with regular expressions for scraping (2 minutes)

# What's wrong with regular expressions for scraping (2 minutes)

- <a href="/whatever/">

# What's wrong with regular expressions for scraping (2 minutes)

- <a href="/whatever/">
- <a href='whatever'>

# What's wrong with regular expressions for scraping (2 minutes)

- <a href="/whatever/">
- <a href='whatever'>
- <a href=‘whatever”>

# What's wrong with regular expressions for scraping (2 minutes)

- <a href="/whatever/">
- <a href='whatever'>
- <a href=‘whatever”>
- Okay for “Reviews 1-10 of 430”

# What's wrong with regular expressions for scraping (2 minutes)

- <a href="/whatever/">
- <a href='whatever'>
- <a href=‘whatever”>
- Okay for “Reviews 1-10 of 430”
- Kodos: Regular expression GUI (since redemo.py seems unmaintained)

# Inspirational quote: Jon Postel (1 minute)

Robustness principle: "Be conservative in what you do, be liberal in what you accept from others."
– Jon Postel, *Transmission Control Protocol*, RFC 793

# Inspirational quote: Leonard Richardson (1 minute)

"You didn't write that awful page. You're just trying to get some data out of it. Right now, you don't really care what HTML is supposed to look like."
– Leonard Richardson, author of BeautifulSoup

# Outline

# Structured data

# Searching those document trees (8 minutes)

# Searching those document trees (8 minutes)

- BeautifulSoup API
  (examples/tree-builders/beautifulsoup/search.py)

# Searching those document trees (8 minutes)

- BeautifulSoup API
  (examples/tree-builders/beautifulsoup/search.py)
- html5lib creates BeautifulSoup objects (or others)
  (examples/tree-builders/html5lib/search.py)

# Searching those document trees (8 minutes)

- BeautifulSoup API
  (examples/tree-builders/beautifulsoup/search.py)
- html5lib creates BeautifulSoup objects (or others)
  (examples/tree-builders/html5lib/search.py)
- lxml provides XPath
  (examples/tree-builders/lxml/search_xpath.py)

# Searching those document trees (8 minutes)

- BeautifulSoup API (examples/tree-builders/beautifulsoup/search.py)
- html5lib creates BeautifulSoup objects (or others) (examples/tree-builders/html5lib/search.py)
- lxml provides XPath (examples/tree-builders/lxml/search_xpath.py)
- “minimal stable XPath”

# Searching those document trees (8 minutes)

- BeautifulSoup API
  (examples/tree-builders/beautifulsoup/search.py)
- html5lib creates BeautifulSoup objects (or others)
  (examples/tree-builders/html5lib/search.py)
- lxml provides XPath
  (examples/tree-builders/lxml/search_xpath.py)
- "minimal stable XPath"
- lxml provides CSSSelect
  (examples/tree-builders/lxml/search_css.py)

# Model the data (3 minutes)

examples/curry/menu.py
class Entree:

# Model the data (3 minutes)

examples/curry/menu.py
class Entree:

- index

# Model the data (3 minutes)

examples/curry/menu.py
class Entree:

- index
- name

# Model the data (3 minutes)

examples/curry/menu.py
class Entree:

- index
- name
- description

# Model the data (3 minutes)

examples/curry/menu.py
class Entree:

- index
- name
- description
- long_winded_description

# Model the data (3 minutes)

examples/curry/menu.py
class Entree:

- index
- name
- description
- long_winded_description
- price

# New goal for curry: Objectify (5 minutes)

Map the menu to Python objects.

# New goal for curry: Objectify (5 minutes)

Map the menu to Python objects.

- ...play with source in BeautifulSoup

# New goal for curry: Objectify (5 minutes)

Map the menu to Python objects.

- ...play with source in BeautifulSoup
- ...realize this is a text processing problem, not a tag processing problem

# Mini-lesson

# Mini-lesson

- hand-written pages vs.

# Mini-lesson

- hand-written pages vs.
- machine-written pages

# New goal: Scrape Yahoo! finance

examples/tree-builders/beautifulsoup_yfinance.py

# We're done!

right?

# Outline

# Interacting with the web (5 minutes)

# Basic Yahoo! search (hard-coded) (2 minutes)

examples/search/yahoo.py

# Basic Google! search (hard-coded) (2 minutes)

examples/search/google.py

# Basic Google! search (hard-coded) (2 minutes)

examples/search/google.py

- Great code, but broken due to ?

# Something's wrong...

# Outline

# More about HTTP (20 minutes)

# A network trace of an HTTP conversation (10 minutes)

# User-Agent, and other headers the client sends (3 minutes)

# Status codes (3 minutes)

# Status codes (3 minutes)

- 2xx: Success

# Status codes (3 minutes)

- 2xx: Success
- 3xx: Redirection

# Status codes (3 minutes)

- 2xx: Success
- 3xx: Redirection
- 4xx: Error

# Status codes (3 minutes)

- 2xx: Success
- 3xx: Redirection
- 4xx: Error
- 402: Payment Required

# Status codes (3 minutes)

- 2xx: Success
- 3xx: Redirection
- 4xx: Error
- 402: Payment Required
- 404 Not Found

# Status codes (3 minutes)

- 2xx: Success
- 3xx: Redirection
- 4xx: Error
- 402: Payment Required
- 404 Not Found
- 410 Gone

# Status codes (3 minutes)

- 2xx: Success
- 3xx: Redirection
- 4xx: Error
- 402: Payment Required
- 404 Not Found
- 410 Gone
- 418 I'm a teapot

# HTTP methods (3 minutes)

# HTTP methods (3 minutes)

- GET

# HTTP methods (3 minutes)

- GET
- POST

# HTTP methods (3 minutes)

- GET
- POST
- PUT

# HTTP methods (3 minutes)

- GET
- POST
- PUT
- BREW

# Once we set User-Agent, are we just like Firefox? (2 minutes)

# Once we set User-Agent, are we just like Firefox? (2 minutes)

- JavaScript behavior

# Once we set User-Agent, are we just like Firefox? (2 minutes)

- JavaScript behavior
- Image download behavior

# Once we set User-Agent, are we just like Firefox? (2 minutes)

- JavaScript behavior
- Image download behavior
- Cookie behavior

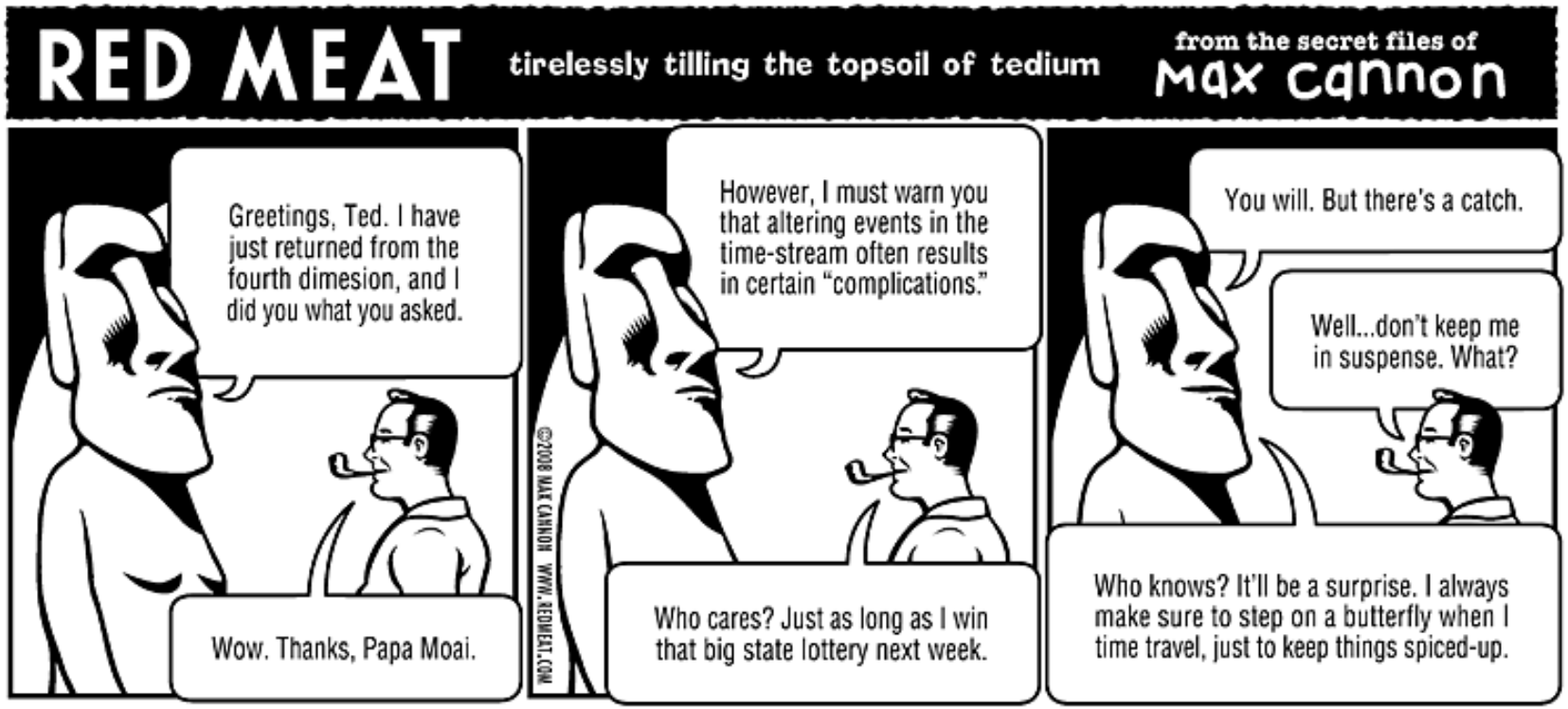
# Once we set User-Agent, are we just like Firefox? (2 minutes)

- JavaScript behavior
- Image download behavior
- Cookie behavior
- Invalid HTML handling behavior (?)

# Once we set User-Agent, are we just like Firefox? (2 minutes)

- JavaScript behavior
- Image download behavior
- Cookie behavior
- Invalid HTML handling behavior (?)
- Accept: headers

# What if we settle for approximate emulation?



# Re-do of Google search with a cooked user-agent (2 minutes)

examples/search/urllib2-user-agent/google_as_ie.py

# Re-do of Google search with a cooked user-agent (2 minutes)

examples/search/urllib2-user-agent/google_as_ie.py

- My favorite user-agent is Internet Explorer 5.0 on Windows 98.

# HTTP: State via cookies (3 minutes)

# HTTP: State via cookies (3 minutes)

- HTTP implements state on *top* of TCP

# robots.txt (2 minutes)

# robots.txt (2 minutes)

- User-agent: *

# robots.txt (2 minutes)

- User-agent: *
- Disallow: /

# robots.txt (2 minutes)

- User-agent: *
- Disallow: /
- Allow: /crawlme.html

# robots.txt (2 minutes)

- User-agent: *
- Disallow: /
- Allow: /crawlme.html
- http://www.robotstxt.org/

# robots.txt and detectability (2 minutes)

# robots.txt and detectability (2 minutes)

- “How does the server know you’re a robot?”

# robots.txt and detectability (2 minutes)

- “How does the server know you’re a robot?”
- Well, if you GET /robots.txt...

# Outline

# Filling out more forms: POST and GET (20 minutes)

(Be sure to pay attention to the clock; minute 90 is when snack break starts.)

# POST: Cepstral Weather demo (by hand) (10 minutes)

http://cepstral.com/cgi-bin/demos/weather

# Note the URL we POST to

# Note the URL we POST to

- from FireBug

# Note the data we POST

# Note the data we POST

- from FireBug

# Write simple Python that also POSTs

examples/cepstral/just_post.py

# Pull out the .wav file and play it with mplayer

examples/cepstral/play_wav.py

# POST: Cepstral weather demo (via mechanize) (3 minutes)

examples/cepstral/just_post_via_mechanize.py

# Basic Yahoo! search (via mechanize) (2 minutes)

examples/search/yahoo_mechanize.py

# Basic Yahoo! search (via mechanize) (2 minutes)

examples/search/yahoo_mechanize.py

- Great code, but broken due to robots.txt

# Basic Yahoo! search (via mechanize, handle_robots=False) (2 minutes)

examples/search/yahoo_mechanize_norobots.py

# Basic Google! search (via mechanize, handle_robots=False, changeuser-agent) (2 minutes)

examples/search/google_mechanize.py

# Outline

# Cookies (10 minutes)

# emusic: Log in and verify that we logged in successfully (with cookielib)(optional) (5 minutes)

examples/cookies/emusic_login_byhand.py

# emusic: Log in and verify that we logged in successfully (with mechanize)(5 minutes)

examples/cookies/emusic_login_mechanize.py

# emusic: Check how many downloads we have left (with mechanize) (5minutes)

examples/cookies/emusic_check_downloads.py

# Now we're done, right?

Whew.

# Outline

# Recap and philosophy (3 minutes)

# Recap

We've seen:

# Recap

We've seen:

- Loading web pages from the network with urllib2

# Recap

We've seen:

- Loading web pages from the network with urllib2
- Parsing web pages (even broken ones)

# Recap

We've seen:

- Loading web pages from the network with urllib2
- Parsing web pages (even broken ones)
- Scraping that page into a set of structured Python objects

# Recap

We've seen:

- Loading web pages from the network with urllib2
- Parsing web pages (even broken ones)
- Scraping that page into a set of structured Python objects
- HTTP status codes

# Recap

We've seen:

- Loading web pages from the network with urllib2
- Parsing web pages (even broken ones)
- Scraping that page into a set of structured Python objects
- HTTP status codes
- Faking the user agent header

# Recap

We've seen:

- Loading web pages from the network with urllib2
- Parsing web pages (even broken ones)
- Scraping that page into a set of structured Python objects
- HTTP status codes
- Faking the user agent header
- Submitting forms

# Recap

We've seen:

- Loading web pages from the network with urllib2
- Parsing web pages (even broken ones)
- Scraping that page into a set of structured Python objects
- HTTP status codes
- Faking the user agent header
- Submitting forms
- Keeping a session with cookies

# “Play nice” on the web (5 minutes)

# “Play nice” on the web (5 minutes)

- Ignore Terms of Service at your own peril

# “Play nice” on the web (5 minutes)

- Ignore Terms of Service at your own peril
- robots.txt

# “Play nice” on the web (5 minutes)

- Ignore Terms of Service at your own peril
- robots.txt
- DO NOT BECOME AN EVIL COMMENT SPAMMER

# Why scrape the web? (4 minutes)

# Why scrape the web? (4 minutes)

- Anger

# Why scrape the web? (4 minutes)

- Anger
- Interoperation with unmaintained systems

# Why scrape the web? (4 minutes)

- Anger
- Interoperation with unmaintained systems
- “Rogue interoperability”

# Three perspectives on web scraping (5 minutes)

# Three perspectives on web scraping (5 minutes)

- Be a Maverick

# Three perspectives on web scraping (5 minutes)

- Be a Maverick
- Use the old standby XML tools

# Three perspectives on web scraping (5 minutes)

- Be a Maverick
- Use the old standby XML tools
- “Use the web tools against the web”

# Outline

# Parser redux

# Choosing a parser

# Choosing a parser

- Performance

# Choosing a parser

- Performance
- Ease-of-use

# Choosing a parser

- Performance
- Ease-of-use
- Quality

# Choosing a parser

- Performance
- Ease-of-use
- Quality
  - Especially as relates to cleaning broken HTML

# Benchmarks by Ian Bicking

# Ease of use

# Quality of resulting tree

# Quality of resulting tree

- lxml $>$ BeautifulSoup

# Quality of resulting tree

- lxml $>$ BeautifulSoup
- lxml $\approx$ html5lib

# Quality of resulting tree

- lxml $>$ BeautifulSoup
- lxml $\approx$ html5lib
- BeautifulSoup 3.0.7 $>$ BeautifulSoup 3.1.0

# A winner

# A winner

- lxml!

# Outline

# Countermeasures

# Imagine a really stupid bot

# Check Referer header

# Check Referer header

- mechanize solves this

# Extra hidden form fields

# Extra hidden form fields

- mechanize solves this

# Requiring cookies

# Requiring cookies

- mechanize solves this

# Outline

# Countermeasures: hard (12 minutes)

# Per-IP address query limits (6 minutes)

Example: Yahoo web search API

# Per-IP address query limits (6 minutes)

Example: Yahoo web search API

- Use more IPs

# Per-IP address query limits (6 minutes)

Example: Yahoo web search API

- Use more IPs
  - Tor, or

# Per-IP address query limits (6 minutes)

Example: Yahoo web search API

- Use more IPs
  - Tor, or
  - your own machines

# Per-IP address query limits (6 minutes)

Example: Yahoo web search API

- Use more IPs
  - Tor, or
  - your own machines
- Use SOCKS (plus SSH) to make this easy

# CAPTCHAs (3 minutes)

Example: Google web search (when you exceed undeclared query limits).

# CAPTCHAs (3 minutes)

Example: Google web search (when you exceed undeclared query limits).

- uh-oh

# JavaScript (3 minutes)

Example: “Hash cash” system for avoiding comment spam.

# JavaScript (3 minutes)

Example: “Hash cash” system for avoiding comment spam.

- uh-oh

# Outline

# Invisible coubntermeasures

# Behavior profiling

# Behavior profiling

- Time-based?

# Inserting false link visible only to bots

# Inserting false link visible only to bots

- “Tarpits”

# robots.txt access

# robots.txt access

- As soon as you access it, you lose.

# Outline

# Getting around IP address limits

# Understand

# Understand

- We still have to stay within the limits. We can just take advantage of IPs we do have.

# ssh -D

# ssh -D

- Borrow the IP of any machine you can log in to

# ssh -D

- Borrow the IP of any machine you can log in to
- ssh -D 1080 asheesh.org

# socks_monkey

# socks_monkey

- SOCKSify Python from within Python

# socks_monkey

- SOCKSify Python from within Python
- examples/ip-limits/socks_monkey.py

# tsocks

# tsocks

- SOCKSify Python via LD_PRELOAD

# tsocks

- SOCKSify Python via LD_PRELOAD
- examples/ip-limits/tsocks/

# tor

"The onion router"

# tor

"The onion router"

- SOCKSify but borrow someone else's IP

# tor

"The onion router"

- SOCKSify but borrow someone else's IP
- (play nice...)

# Cycling strategies

# Cycling strategies

- Drain it dry

# Cycling strategies

- Drain it dry
  - easy to implement first

# Cycling strategies

- Drain it dry
  - easy to implement first
- Round-robin

# Cycling strategies

- Drain it dry
  - easy to implement first
- Round-robin
  - generally preferable

# Outline

# JavaScript: breaking Hash Cash (15 minutes)

# Detecting its presence

# Detecting its presence

- Attempt to submit a comment with JS disabled

# Detecting its presence

- Attempt to submit a comment with JS disabled
- Attempt to submit a comment with JS enabled

# Detecting its presence

- Attempt to submit a comment with JS disabled
- Attempt to submit a comment with JS enabled
- Trace the second in FireBug

# Rewriting the JavaScript as Python

# Rewriting the JavaScript as Python

- You may think I'm joking, but this is a common strategy.

# DOMForm

# DOMForm

- Good news

"DOMForm is a Python module for web scraping and web testing. It knows how to evaluate embedded JavaScript code in response to appropriate events."
– John J. Lee of mechanize

# DOMForm

- Good news

"DOMForm is a Python module for web scraping and web testing. It knows how to evaluate embedded JavaScript code in response to appropriate events."
– John J. Lee of mechanize

- Bad news

"This module is unmaintained. Maybe someday..."
Also, it does not execute page-global JavaScript, which is where HashCash is implemented.

# python-spidermonkey

# python-spidermonkey

- Good news

# python-spidermonkey

- Good news
  - “Python/JavaScript bridge module, making use of Mozilla’s spidermonkey JavaScript implementation.”

# python-spidermonkey

- Good news
  - "Python/JavaScript bridge module, making use of Mozilla's spidermonkey JavaScript implementation."
- Bad news

# python-spidermonkey

- Good news
  - "Python/JavaScript bridge module, making use of Mozilla's spidermonkey JavaScript implementation."
- Bad news
  - ...do you really want to parse the web page for JavaScript and execute it?

# python-spidermonkey

- Good news
  - "Python/JavaScript bridge module, making use of Mozilla's spidermonkey JavaScript implementation."
- Bad news
  - ...do you really want to parse the web page for JavaScript and execute it?
  - examples/javascript/hashcash.py

# lck

# Ick

- None of this is as clean and automated as mechanize.

# Outline

# “Breaking” CAPTCHAs (15 minutes)

# Fallback: yourself

# Fallback: yourself

- Can always just prompt the operator to figure it out and enter it

# Mailinator: “Enter these words to delete the email”

# Mailinator: “Enter these words to delete the email”

- Only so many different images

# Mailinator: “Enter these words to delete the email”

- Only so many different images
- So build a look-up table

# Mailinator: “Enter these words to delete the email”

- Only so many different images
- So build a look-up table
- ...indexed by URL?

# Mailinator: “Enter these words to delete the email”

- Only so many different images
- So build a look-up table
- ...indexed by URL?
- ...indexed by image contents?

# Mailinator: “Enter these words to delete the email”

- Only so many different images
- So build a look-up table
- ...indexed by URL?
- ...indexed by image contents?
- ...indexed by fuzzy image contents?

(I don’t have a good tool for the last one.)

# Audio captchas: “Simple” signal analysis

# Audio captchas: “Simple” signal analysis

- Should be doable in pylab/matplotlib with fast Fourier transforms

# JavaScript CAPTCHAs (like reCAPTCHA)

# JavaScript CAPTCHAs (like reCAPTCHA)

- re-implement CAPTCHA-downloading logic in Python

# JavaScript CAPTCHAs (like reCAPTCHA)

- re-implement CAPTCHA-downloading logic in Python
- ...or execute the JavaScript with spidermonkey

# Outline

# The website from Hell: US PTO Public PAIR

http://portal.uspto.gov/external/portal/pair

# Start with a CAPTCHA

# Solve it and move on to...

# Solve it and move on to...

- document.write()

The page is invisible.

# Outline

# Automating the web browser

# Selenium Remote Control (5 minutes)

examples/seleniumrc/start.py

# Selenium IDE (10 minutes)

# Selenium IDE (10 minutes)

- Our friend, XPath

# Selenium IDE (10 minutes)

- Our friend, XPath
- FireBug

# Why don't we just do this all the time?

# Why don't we just do this all the time?

- Firefox memory footprint

# Why don't we just do this all the time?

- Firefox memory footprint
- Flexibility

# Outline

# Conclusions

# Scaling and stability (5 minutes)

# Scaling and stability (5 minutes)

- Choosing reliable queries from web pages

# Scaling and stability (5 minutes)

- Choosing reliable queries from web pages
- Expanding to more IP addresses when necessary using SSH (and Python 2.6 multiprocessing for a plausible model of how to rotate SOCKS proxies)

# Scaling and stability (5 minutes)

- Choosing reliable queries from web pages
- Expanding to more IP addresses when necessary using SSH (and Python 2.6 multiprocessing for a plausible model of how to rotate SOCKS proxies)
- Tor (and other proxy considerations)

# Scaling and stability (5 minutes)

- Choosing reliable queries from web pages
- Expanding to more IP addresses when necessary using SSH (and Python 2.6 multiprocessing for a plausible model of how to rotate SOCKS proxies)
- Tor (and other proxy considerations)
- registrar.py: Six and a half years stable

# Summary (5 minutes)

# Summary (5 minutes)

- If it's on a web page, you can scrape it out.

# Summary (5 minutes)

- If it's on a web page, you can scrape it out.
- "Now you have an API for everything."

# Future directions

# Future directions

- More automation

# Future directions

- More automation
  - Automatic migration from data tables into Python dictionaries

# Future directions

- More automation
  - Automatic migration from data tables into Python dictionaries
  - Crawling a site to verify assumptions

# Future directions

- More automation
  - Automatic migration from data tables into Python dictionaries
  - Crawling a site to verify assumptions
- Do you do these?

# Bonus time (? minutes)

If we have time:

# Bonus time (? minutes)

If we have time:

- Greasemonkey demo: scraping in the browser

# Bonus time (? minutes)

If we have time:

- Greasemonkey demo: scraping in the browser
- Audience-suggested scraping lab

# Bonus time (? minutes)

If we have time:

- Greasemonkey demo: scraping in the browser
- Audience-suggested scraping lab
- Workshopping on queries or regular expressions