14.170: Programming for Economists

1/12/2009-1/16/2009

Melissa Dell

Matt Notowidigdo

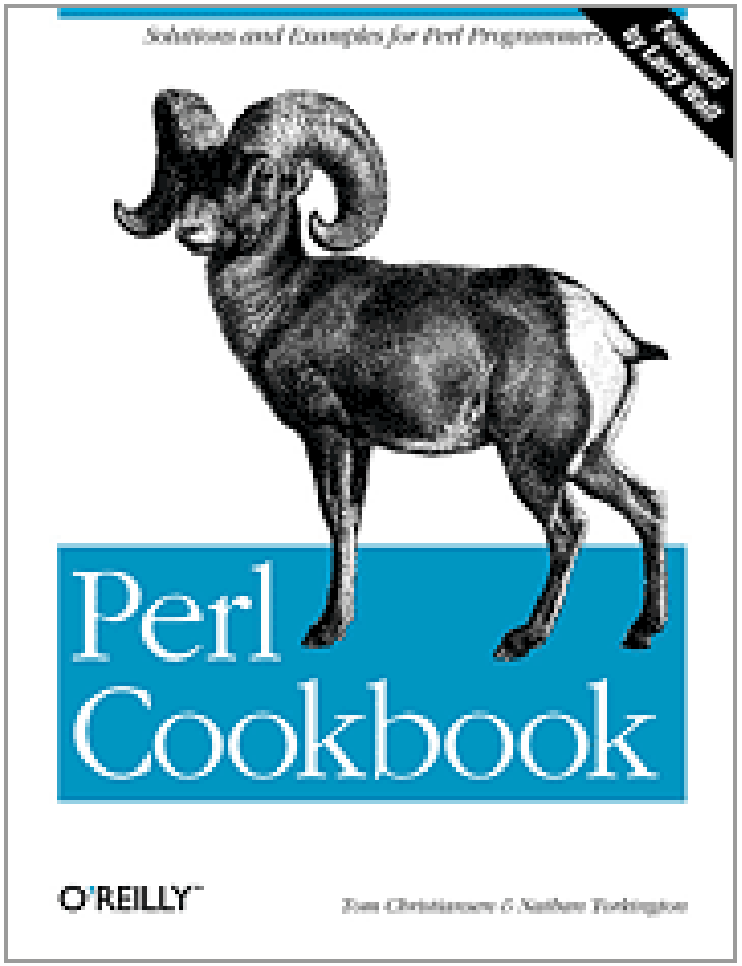
Paul Schrimpf

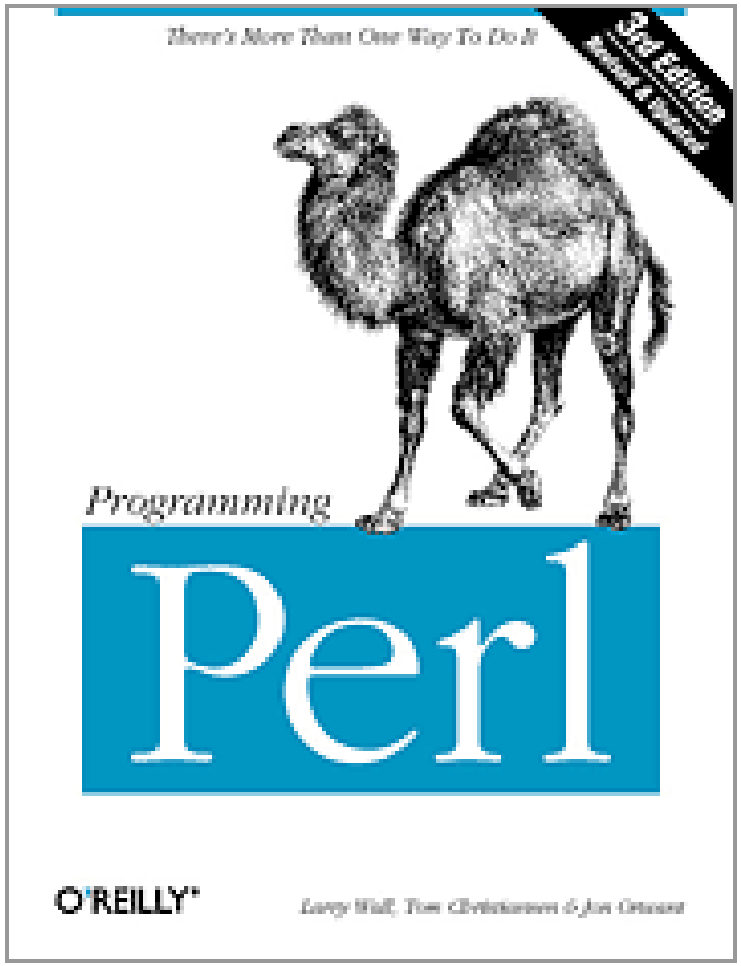
Perl (for economists)

Perl overview slide

- This short lecture will go over what I feel are the primary uses of Perl (by economists)
  - To use Perl's built-in data structures to implement algorithms with asymptotically superior runtime (as compared to Stata/Matlab)
  - Web crawlers to automatically download data. At MIT, I know Paul Schrimpf, Tal Gross, Tom Chang, Mar Reguant Ridó and I have all used Perl for this purpose
    - Web crawlers also used in Ellison & Ellison, Shapiro & Gentzkow, Greg Lewis job market paper, Price and Wolfers).
  - To parse structured text for the purposes of creating a dataset (oftentimes, after that dataset was downloaded by a web crawler)

Where to learn Perl





Today’s goals

- Learn how to run Perl
- Learn basic Perl syntax
- Learn about hash tables
- See example code doing each of the following:
  - Preparing data
  - Downloading data
  - Parsing data

How to run Perl

- In theory, Perl is “cross-platform”. You can “write [it] once, run [it] anywhere.” In practice, Perl is usually run on UNIX or Linux. In the econ computer cluster, you can’t install Perl on Windows machines because they are a (perceived) security risk.
- So in econ cluster you will have to run on UNIX/Linux using “SecureCRT” or some other terminal emulator.
  - Alternatively, you can go to Athena cluster in basement of E51 and run Perl on the Athena computer
- Perl is installed on every UNIX/Linux machine by default.

How to run Perl, con't

- SSH into UNIX server blackmarket/shadydealings/etc. (open TWO windows, one window for writing code, one window for running the code)
- Use emacs (or some other text editor) to edit the Perl file. Make sure the suffix of the file is ".pl" and then you can run the file by typing "perl myfile.pl" at the command line
- To start emacs, type "emacs myfile.pl" and "myfile.pl" will be created (click "tools" on 14.170 course webpage where there is a nice emacs introduction). It's worth learning emacs if you will be writing a lot of Perl code

How to run Perl, con’t

blackmarket.mit.edu - SecureCRT

blackmarket.mit.edu

```
File Edit Options Buffers Tools Help
#!/usr/bin/perl

$name = $ARGV[0];
$age  = $ARGV[1];

print "Hello $name!  You are $age years old!\n\n";

-uuu:---F1  myfile.pl          (Perl)--L6--All--------------------
Wrote /bbkinghome/noto/myfile.pl
```

Ready | ssh2: AES-128 | 7, 49 | 28 Rows, 65 Cols | VT100

blackmarket.mit.edu - SecureCRT

blackmarket.mit.edu

```
[noto@blackmarket ~]$ perl myfile.pl "Matt Notowidigdo" 26
Hello Matt Notowidigdo!  You are 26 years old!

[noto@blackmarket ~]$
```

Ready | ssh2: AES-128 | 4, 23 | 29 Rows, 63 Cols | VT100

Basic Perl syntax

- 3 types of variables:
  - scalars
  - arrays
  - hash tables

- They are created using different characters:
  - scalars are created as `$scalar`
  - arrays are created as `@array`
  - hash tables are created as `%hashtable`

- So the `$ @ %` characters tell Perl what is the TYPE of the variable. This is obviously not very clear syntax. In Java, for example, here is how you create an array and a hash table:

```
ArrayList myarray     = new ArrayList();
Hashtable myhashtable = new Hashtable();
```

- In Perl the same code is the following:

```
@mylist      = ();
%myhashtable = ();
```

Hello World!

```
#!/usr/bin/perl
$hello1 = "Hello World!\n";
$econ = 14;
@hello2 = ("Hello World!\n",
           "Hello World again!\n");
print $hello1;
print $hello2[0];
print $hello2[1];
print $econ;
```

blackmarket.mit.edu

```
[noto@blackmarket ~/14.170]$ perl myfile.pl
Hello World!
Hello World!
Hello World again!
14
[noto@blackmarket ~/14.170]$
```

Control structures

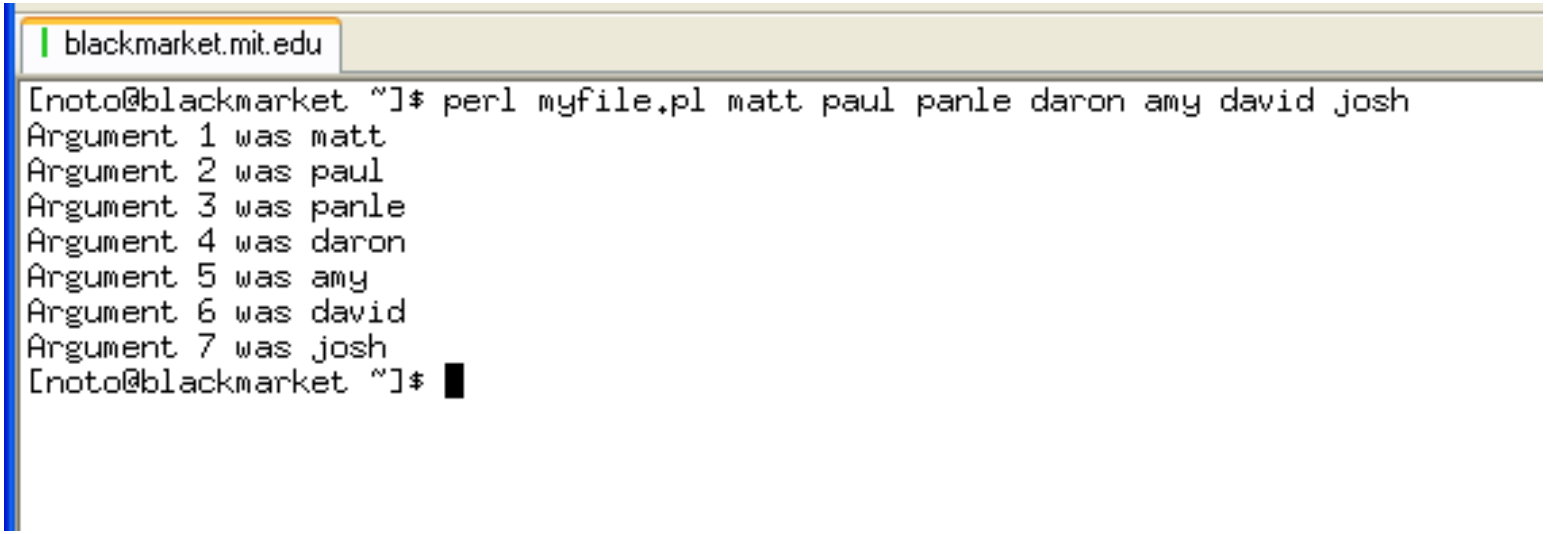
```
#!/usr/bin/perl
$top = $ARGV[0];
for ($i = 1; $i < $top; $i++) {
  if ( int($i / 7) == ($i / 7) ) {
      print "$i is a multiple of 7!\n";
  }
}
```

blackmarket.mit.edu

```
[noto@blackmarket ~]$ perl myfile.pl 105
7 is a multiple of 7!
14 is a multiple of 7!
21 is a multiple of 7!
28 is a multiple of 7!
35 is a multiple of 7!
42 is a multiple of 7!
49 is a multiple of 7!
56 is a multiple of 7!
63 is a multiple of 7!
70 is a multiple of 7!
77 is a multiple of 7!
84 is a multiple of 7!
91 is a multiple of 7!
98 is a multiple of 7!
[noto@blackmarket ~]$
```

@ARGV

```
#!/usr/bin/perl
$i=1;
foreach $arg (@ARGV) {
  print "Argument $i was $arg \n";
  $i+=1;
}
```



Regular expressions

```
#!/usr/bin/perl
foreach $arg (@ARGV) {
  if ($arg =~ /^perl/) {
      print "The word $arg starts with perl!\n";
  }
}
```

```
blackmarket.mit.edu
[noto@blackmarket ~]$ perl myfile.pl perlocution pearl perlite perleche
The word perlocution starts with perl!
The word perlite starts with perl!
The word perleche starts with perl!
[noto@blackmarket ~]$
```

Regular expressions, con't

```
#!/usr/bin/perl
foreach $arg (@ARGV) {
  if ($arg =~ /^([a-zA-Z]+)$/) {
      print "The argument $arg contains only characters!\n";
  }
  else {
      if ($arg =~ /^([a-zA-Z0-9]+)$/) {
          print "The argument $arg contains only numbers and characters!\n";
      }
      else {
          print "The argument $arg contains non-alphanumeric characters!\n";
      }
  }
}
```

blackmarket.mit.edu

```
[noto@blackmarket ~]$ perl myfile.pl noto gr8 panle hot4u paul_s 2good4me
The argument noto contains only characters!
The argument gr8 contains only numbers and characters!
The argument panle contains only characters!
The argument hot4u contains only numbers and characters!
The argument paul_s contains non-alphanumeric characters!
The argument 2good4me contains only numbers and characters!
[noto@blackmarket ~]$
```

Regular expressions, con't

```
#!/usr/bin/perl
foreach $arg (@ARGV) {
  if ($arg =~ /^\d\d\d\-\d\d\d\-\d\d\d\d$/) {
      print "$arg is a valid phone number!\n";
  }
  else {
      print "$arg is an invalid phone number!\n";
  }
}
```

blackmarket.mit.edu

```
[noto@blackmarket ~]$ perl myfile.pl "(111)-111-1111" "222-222-2222" "333-3333" "444-4444444" "ABC-DEF-GHIJ"
(111)-111-1111 is an invalid phone number!
222-222-2222 is a valid phone number!
333-3333 is an invalid phone number!
444-4444444 is an invalid phone number!
ABC-DEF-GHIJ is an invalid phone number!
[noto@blackmarket ~]$
```

Regular expressions, con't

```
#!/usr/bin/perl
foreach $arg (@ARGV) {
  if ($arg =~ /^(\d{3})-(\d{3})-(\d{4})$/) {
      print "$arg is a valid phone number!\n";
  }
  else {
      print "$arg is an invalid phone number!\n";
  }
}
```

blackmarket.mit.edu

```
[noto@blackmarket ~]$ perl myfile.pl "(111)-111-1111" "222-222-2222" "333-3333" "444-4444444" "ABC-DEF-GHIJ"
(111)-111-1111 is an invalid phone number!
222-222-2222 is a valid phone number!
333-3333 is an invalid phone number!
444-4444444 is an invalid phone number!
ABC-DEF-GHIJ is an invalid phone number!
[noto@blackmarket ~]$
```

Regular expressions, con't

```
#!/usr/bin/perl
foreach $arg (@ARGV) {
  if ($arg =~ /^(\d{3})-(\d{3})-(\d{4})$/) {
      print "$arg is a valid phone number!\n";
      print "  area code: $1 \n";
      print "  number:    $2-$3 \n";
  }
  else {
      print "$arg is an invalid phone number!\n";
  }
}
```

blackmarket.mit.edu

```
[noto@blackmarket ~]$ perl myfile.pl "(111)-111-1111" "222-222-2222" "333-3333" "444-4444444" "ABC-DEF-GHIJ"
(111)-111-1111 is an invalid phone number!
222-222-2222 is a valid phone number!
  area code: 222
  number:    222-2222
333-3333 is an invalid phone number!
444-4444444 is an invalid phone number!
ABC-DEF-GHIJ is an invalid phone number!
[noto@blackmarket ~]$
```

Regular expressions, con't

```
#!/usr/bin/perl
foreach $arg (@ARGV) {
  if ($arg =~ /^\(?(\d{3})\)?-(\d{3})-(\d{4})$/) {
      print "$arg is a valid phone number!\n";
      print "  area code: $1 \n";
      print "  number:    $2-$3 \n";
  }
  else {
      print "$arg is an invalid phone number!\n";
  }
}
```

```
blackmarket.mit.edu
[noto@blackmarket ~]$ perl myfile.pl "(111)-111-1111" "222-222-2222" "333-3333" "444-4444444" "ABC-DEF-GHIJ"
(111)-111-1111 is a valid phone number!
  area code: 111
  number:    111-1111
222-222-2222 is a valid phone number!
  area code: 222
  number:    222-2222
333-3333 is an invalid phone number!
444-4444444 is an invalid phone number!
ABC-DEF-GHIJ is an invalid phone number!
[noto@blackmarket ~]$
```

Regular expressions, con't

```
#!/usr/bin/perl
foreach $arg (@ARGV) {
  if ($arg =~ /^\(?(\d{3})\)?-(\d{3})-?(\d{4})$/) {
      print "$arg is a valid phone number!\n";
      print "  area code: $1 \n";
      print "  number:    $2-$3 \n";
  }
  else {
      print "$arg is an invalid phone number!\n";
  }
}
```

```
blackmarket.mit.edu
[noto@blackmarket ~]$ perl myfile.pl "(111)-111-1111" "222-222-2222" "333-3333" "444-4444444" "ABC-DEF-GHIJ"
(111)-111-1111 is a valid phone number!
  area code: 111
  number:    111-1111
222-222-2222 is a valid phone number!
  area code: 222
  number:    222-2222
333-3333 is an invalid phone number!
444-4444444 is a valid phone number!
  area code: 444
  number:    444-4444
ABC-DEF-GHIJ is an invalid phone number!
[noto@blackmarket ~]$
```

Regular expressions, con't

```
#!/usr/bin/perl
foreach $arg (@ARGV) {
  if ($arg =~ /^(\(?(\d{3})\)?)?-?(\d{3})-?(\d{4})$/) {
      print "$arg is a valid phone number!\n";
      print "  area code: " . ($2 eq "" ? "unknown" : $2) . " \n";
      print "  number:    $3-$4 \n";
  }
  else {
      print "$arg is an invalid phone number!\n";
  }
}
```

```
blackmarket.mit.edu
[noto@blackmarket ~]$ perl myfile.pl "(111)-111-1111" "222-222-2222" "333-3333" "444-4444444" "ABC-DEF-GHIJ"
(111)-111-1111 is a valid phone number!
  area code: 111
  number:    111-1111
222-222-2222 is a valid phone number!
  area code: 222
  number:    222-2222
333-3333 is a valid phone number!
  area code: unknown
  number:    333-3333
444-4444444 is a valid phone number!
  area code: 444
  number:    444-4444
ABC-DEF-GHIJ is an invalid phone number!
[noto@blackmarket ~]$
```

Regular expressions, con't

```
#!/usr/bin/perl
foreach $arg (@ARGV) {
  if ($arg =~ /^(\(?(\d{3})\)?)?-?(\d{3})-?(\d{4})$/) {
      print "$arg is a valid phone number!\n";
      print "  area code: " . ($2 eq "" ? "unknown" : $2) . " \n";
      print "  number:    $3-$4 \n";
  }
  else {
      print "$arg is an invalid phone number!\n";
  }
}
```

QUIZ:

What would happen to the following patterns?

"5555555555"

"(666)666-6666"

"(777)-7777777"

Regular expressions, con't

```
#!/usr/bin/perl
foreach $arg (@ARGV) {
  if ($arg =~ /^(\(?(\d{3})\)?)?-?(\d{3})-?(\d{4})$/) {
      print "$arg is a valid phone number!\n";
      print "  area code: " . ($2 eq "" ? "unknown" : $2) . " \n";
      print "  number:    $3-$4 \n";
  }
  else {
      print "$arg is an invalid phone number!\n";
  }
}
```

QUIZ:

What would happen to the following patterns?

"5555555555"

"(666)666-6666"

"(777)-7777777"

```
blackmarket.mit.edu
[noto@blackmarket ~]$ perl myfile.pl "5555555555" "(666)666-6666" "(777)-7777777"
5555555555 is a valid phone number!
  area code: 555
  number:    555-5555
(666)666-6666 is a valid phone number!
  area code: 666
  number:    666-6666
(777)-7777777 is a valid phone number!
  area code: 777
  number:    777-7777
[noto@blackmarket ~]$
```

Regular expressions, con't

```
#!/usr/bin/perl
foreach $arg (@ARGV) {
  if ($arg =~ /^(\(?(\d{3})\)?)?-?(\d{3})-?(\d{4})$/) {
      print "$arg is a valid phone number!\n";
      print "  area code: " . ($2 eq "" ? "unknown" : $2) . " \n";
      print "  number:    $3-$4 \n";
  }
  else {
      print "$arg is an invalid phone number!\n";
  }
}
```

QUIZ:

What would happen to the following patterns?

"(5555555555"

"666)-666-6666"

Regular expressions, con't

```
#!/usr/bin/perl
foreach $arg (@ARGV) {
  if ($arg =~ /^(\(?(\d{3})\)?)?-?(\d{3})-?(\d{4})$/) {
      print "$arg is a valid phone number!\n";
      print "  area code: " . ($2 eq "" ? "unknown" : $2) . " \n";
      print "  number:    $3-$4 \n";
  }
  else {
      print "$arg is an invalid phone number!\n";
  }
}
```

QUIZ:

What would happen to the following patterns?

"(5555555555"

"666)-666-6666"

bootlegger.mit.edu

```
[noto@bootlegger ~/14.170]$ perl test.pl "(555555-5555"
(555555-5555 is a valid phone number!
  area code: 555
  number:    555-5555
[noto@bootlegger ~/14.170]$ perl test.pl "666)-666-6666"
666)-666-6666 is a valid phone number!
  area code: 666
  number:    666-6666
[noto@bootlegger ~/14.170]$
```

Parsing HTML

```
#!/usr/bin/perl
foreach $arg (@ARGV) {
    if ($arg =~ /^<tr><td>(.*)<\/td><td>(.*)<\/td><\/tr>$/) {
        print "data: $1, $2\n";
    }
}
```

```
blackmarket.mit.edu
[noto@blackmarket ~]$ perl myfile.pl "<tr><td>14</td><td>0.25</td></tr>"
data: 14, 0.25
[noto@blackmarket ~]$
```

Detroit Pistons vs Cleveland Cavaliers Eastern Conference Finals Home Game 3 (If Necessary) Tic - Windows Internet Explorer

http://www.aceticket.com/event/565114

File Edit View Favorites Tools Help

Links

Detroit Pistons vs Cleveland Cavaliers Eastern Confer...

Page Tools

Buy and Sell by phone 1-800-MY-SEATS

Red Sox • Bruins • Celtics • Patriots • BC Eagles • Boston Events • Sports • Concerts • Theatre • Information • View Cart

SEARCH FOR TICKETS | HOME | BUY TICKETS | SELL TICKETS | Today is May 28, 2007

Event

State

--Select a State--

Need more search options?

Find

GOT A QUESTION?
ASK LIVE!

>> OFFLINE

SPORTS TICKETS
- MLB Baseball
- NFL Football
- NHL Hockey
- NBA Basketball
- College Sports
- NASCAR
- WFL Fighting

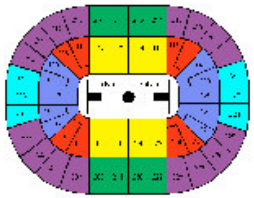
NEW CONCERTS!
- Nickelback
- Kelly Clarkson
- Tool
- Poison
- Justin Timberlake
- John Mayer
- Rush
- American Idols Live
- Stevie Nicks
- Allman Brothers Band
- Wilco

Detroit Pistons vs Cleveland Cavaliers Eastern Conference Finals Home Game 3 (If Necessary) Tickets

Ace Ticket (1-800-MY-SEATS) - Your #1 Ticket Source!

Detroit Pistons vs Cleveland Cavaliers Eastern Conference Finals Home Game 3 (If Necessary)
May 31, 2007 8:00 PM
Palace of Auburn Hills (Basketball) (Auburn Hills, MI)

If you're looking for a particular quantity of tickets that are not shown here or if you simply can't find what you're looking for give us a call. We often have additional inventory that may not be listed on our website. Call 800-697-3287 and we'll be happy to accommodate any special requests or general questions you may have.



click to enlarge map

SECTION / ROW	PRICE	QTY	
210 ROW 13 ROUND 3 HG 3 TICKETFAST	$85.00	8	Add To Cart
223 ROW 04 ROUND 3 HG 3 TICKETFAST	$90.00	8	Add To Cart
223 ROW 6 EMAIL DELIVERY AVAILABLE!! -- (Round 3--Home Game 3)	$90.00	8	Add To Cart
222 ROW 9 Round 3 Home Game 3	$90.00	2	Add To Cart
210 ROW 13 HOME GAME 3 - UPPER LEVEL BEHIND THE BASKET	$105.00	2	Add To Cart
209 ROW 12 HOME GAME 3 - UPPER LEVEL BEHIND THE BASKET	$105.00	2	Add To Cart

Done | Internet | 100%

```
<tr bgcolor="#EEEEEE" height="45" onmouseover="style.backgroundColor='#E0E0E0';"
    onmouseout="style.backgroundColor='#EEEEEE'"><td class="td_smalltext"
    valign="middle" align="left"><DIV style="border-style:none; padding-left:5px;
    padding-right:5px;"><b>210</b> <img
    src="http://www.aceticket.com/images/transpacer.gif" width="5">ROW 13<br><font
    color="#666666">ROUND 3 HG 3 TICKETFAST</font></div></td>
<td class="td_smalltext" valign="middle" align="center">$85.00</td>
<td class="td_smalltext" valign="middle" align="center" valign="middle"><select
    name="quantity1239322161"><option>8</option><option>6</option><option>4</option><op
    tion>2</option></select></td>
<td class="td_smalltext" valign="middle" align="center"><a href="#" class="link_red"
    onClick="JavaScript: return addToCart('1239322161');"><img
    src=http://www.aceticket.com/images/button_add_to_cart.gif border=0></a></td>
                </tr>
                <tr><td colspan="5"
    background="http://www.aceticket.com/images/dotted_bg.jpg"><img
    src="http://www.aceticket.com/images/transpacer.gif" height="2" /></td></tr>
                <tr bgcolor="#FFFFFF" height="45"
    onmouseover="style.backgroundColor='#E0E0E0';"
    onmouseout="style.backgroundColor='#FFFFFF'">

<td class="td_smalltext" valign="middle" align="left"><DIV style="border-style:none;
    padding-left:5px; padding-right:5px;"><b>223</b> <img
    src="http://www.aceticket.com/images/transpacer.gif" width="5">ROW 04<br><font
    color="#666666">ROUND 3 HG 3 TICKETFAST</font></div></td>
<td class="td_smalltext" valign="middle" align="center">$90.00</td>
<td class="td_smalltext" valign="middle" align="center" valign="middle"><select
    name="quantity1239540186"><option>8</option><option>6</option><option>4</option><op
    tion>2</option></select></td>
<td class="td_smalltext" valign="middle" align="center"><a href="#" class="link_red"
    onClick="JavaScript: return addToCart('1239540186');"><img
    src=http://www.aceticket.com/images/button_add_to_cart.gif border=0></a></td>
                </tr>

   ...
   ...
   ...
```

Parsing HTML

```
## header row in TAB-delimited file
print "ticketId\tsection\tmaxAvailable\tprice\n";

## fields that parser will try to detect
$ticketId = "null";
$price = "null";
$maxAvailable = "null";
$section = "null";

$on = 0;
open(FILE, $ARGV[0]);
while ($line = <FILE>) {
    if ($on eq 0 and $line =~ /<tr/) {  $on = 1;  }
    if ($on eq 1) {
        if ($line =~ /addToCart\(\'(.*?)\'\)/) {  $ticketId = $1;  }
        if ($line =~ /<select(.*?)><option>(.*?)<\/option>/) {
            $maxAvailable = $2;
        }
        if ($line =~ />\$(.*?)</) {   $price = $1;    }
        if ($line =~ /<DIV(.*?)><b>(.*?)<\/b>/) {  $section = $2;   }
        if ($line =~ /<\/tr>/) {
            $on = 0;
            if ($ticketId ne "null") {
                print "$ticketId\t$section\t$maxAvailable\t$price\n";
                $ticketId = "null";
                $price = "null";
                $maxAvailable = "null";
                $section = "null";
            }
        }
    }
}
close(FILE);
```

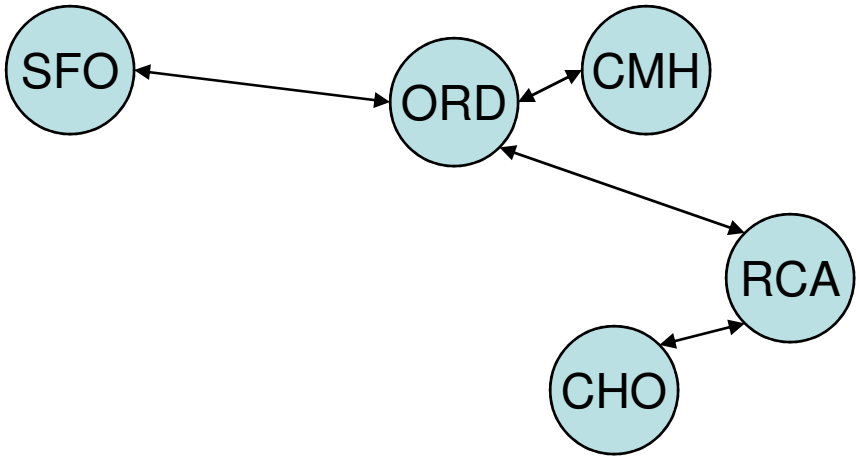
Parsing HTML

bootlegger.mit.edu

```
[noto@bootlegger ~/14.170]$ perl parser.pl data.txt
ticketId        section maxAvailable    price
1239322161      210     8       85.00
1239540186      223     8       90.00
1239110186      221     8       90.00
1200540186      101     8       90.00
1239540996      223     8       90.00
1989540186      223     2       100.00
2039540186      303     8       50.00
[noto@bootlegger ~/14.170]$
[noto@bootlegger ~/14.170]$
[noto@bootlegger ~/14.170]$ perl parser.pl data.txt > parsed_data.txt
[noto@bootlegger ~/14.170]$
[noto@bootlegger ~/14.170]$
```

Using control structures for data preparation

EXAMPLE: Find all 1-city layover flights given data set of available flights



origin	dest	carrier
SFO	ORD	Delta
ORD	SFO	Delta
ORD	CMH	Delta
CMH	ORD	Delta
ORD	RCA	Delta
RCA	ORD	Delta
CHO	RCA	Delta
RCA	CHO	Delta

Hash Tables

Let's go back to Lecture 1 …

<u>LAYOVER BUILDER ALGORITHM</u>

In the raw data, observations are (O, D, C, . , .) tuple where

O = origin

D = destination

C = carrier string

and last two arguments are missing (but will be the second carrier and layover city)

```
FOR each observation i from 1 to N
  FOR each observation j from i+1 to N
    IF D[i] == O[j] & O[i] != D[j]
      CREATE new tuple (O[i], D[j], C[i], C[j], D[i])
```

Hash Tables

Let's loosely prove the runtime …

```
FOR each observation i from 1 to N
  FOR each observation j from i+1 to N
    IF D[i] == O[j] & O[i] != D[j]
      CREATE new tuple (O[i], D[j], C[i], C[j], D[i])
```

First line is done N times. Inside the first loop, there are N – *i* iterations. Assume the last two lines take $O(1)$ time (as they would in Matlab/C). Then total runtime is (N-1 + N-2 + … 2 + 1)*$O(1)$ = $O(0.5*N*(N-1))$ = $O(N^2)$

Hash Tables

Let's imagine augmenting the algorithm as follows:

<u>NEW(!) LAYOVER BUILDER ALGORITHM</u>

```
FOR each observation i from 1 to N
  LIST p = GET all flights that start with D[i]
    FOR each observation j in p
      IF O[i] != D[j]
        CREATE new tuple (O[i], D[j], C[i], C[j], D[i])
```

Hash Tables

What's the runtime here …

```
FOR each observation i from 1 to N
  LIST p = GET all flights that start with D[i]
    FOR each observation j in p
      IF O[i] != D[j]
        CREATE new tuple (O[i], D[j], C[i], C[j], D[i])
```

(LOOSE proof) First line is done N times. Inside the first loop, there is a GET command. Assume that the GET command takes $O(1)$ time. Then there are K iterations in the second FOR loop (where K is number of flights that start with D[i]; assume for simplicity this is constant across all observations). Assume, as before, that the last two lines take $O(1)$ time (as they would in Matlab/C). Then total runtime is $(N*K)*O(1)$ = ***O*(K*N)**

NOTE 1: If K is constant (i.e. doesn't scale with N), then this algorithm is ***O*(N)**. K being constant is not an unreasonable assumption. It means that as you add more origin-destination pairs, the number of flights per airport is constant (i.e. the density of the O-D matrix is constant as N gets larger)

NOTE 2: The "magic" is the $O(1)$ line in the GET command. If that command took $O(N)$ time instead (say, because it had to look through every observation), then the algorithm would be $O(N^2)$ as before. Thus we need a data structure that can return all flights that start with D[i] in constant time. That's what a hash table is used for. Think of a hash table as DICTIONARY. When you want to look up a word in a dictionary, you don't naively look through all the pages, you "sorta know" where you want to start looking.

Hash table syntax

```
#!/usr/bin/perl
foreach $arg (@ARGV) {
    if ($arg =~ /^(.+)=(.+)$/) {
        $hashtable{$1} = $2;
    }
}
print $hashtable{"economics"} . "\n";
print $hashtable{"art history"} . "\n";
print $hashtable{"political science"} . "\n";
print $hashtable{"math"} . "\n";
```

blackmarket.mit.edu

```
[noto@blackmarket ~/14.170]$ perl myfile.pl "math=18" "economics=14" "political science=17"
14

17
18
[noto@blackmarket ~/14.170]$
```

dep_str	arr_str	origin	dest	carrier	dep_mins	arr_mins
2:02 AM	4:45 AM	GBG	SFO	Delta	122	285
7:06 PM	9:43 PM	ORD	SFO	Delta	1146	1303
6:39 AM	8:29 AM	BTR	SFO	Delta	399	509
2:54 PM	5:01 PM	LGA	SFO	Delta	894	1021
1:59 AM	4:52 AM	BTR	SFO	Delta	119	292
7:39 AM	10:21 AM	GBG	SFO	Delta	459	621
2:27 AM	4:54 AM	BBB	SFO	Delta	147	294
2:57 PM	5:46 PM	CHO	SFO	Delta	897	1066
2:57 PM	4:34 PM	DDS	SFO	Delta	897	994
11:12 AM	12:38 PM	LGA	SFO	Delta	672	758
12:37 PM	3:03 PM	QDE	SFO	Delta	757	903
12:29 AM	2:42 AM	QQE	SFO	Delta	29	162
6:17 AM	8:06 AM	JJJ	SFO	Delta	377	486
7:41 AM	9:02 AM	LAS	SFO	Delta	461	542
12:48 AM	3:22 AM	CMH	SFO	Delta	48	202
2:27 PM	4:07 PM	VFB	SFO	Delta	867	967
3:15 AM	4:15 AM	ITH	SFO	Delta	195	255
5:36 PM	7:11 PM	QDE	SFO	Delta	1056	1151
9:26 AM	11:54 AM	ITH	SFO	Delta	566	714
9:43 AM	12:09 PM	MYR	SFO	Delta	583	729
12:15 AM	1:47 AM	VDZ	SFO	Delta	15	107
7:19 PM	9:46 PM	GBG	SFO	Delta	1159	1306
6:51 AM	8:38 AM	YGR	SFO	Delta	411	518
3:11 AM	5:46 AM	BBB	SFO	Delta	191	346
4:58 AM	6:01 AM	QDE	SFO	Delta	298	361
9:19 AM	10:33 AM	LAX	SFO	Delta	559	633
11:14 AM	12:31 PM	JJJ	SFO	Delta	674	751
9:30 AM	12:22 PM	LLL	SFO	Delta	570	742

Old algorithm

```
open(FILE, "air.txt");
$numobs= 0;
$line = <FILE>;
while($line = <FILE>) {
    my @data_line = split(/\t|\n|\r/, $line);
    push(@data, [@data_line] );
    $numobs++;
}
close(FILE);

for ($i = 0; $i < $numobs; $i++) {
    for ($j = 0; $j < $numobs; $j++) {
        if ($data[$i][6] + 45 < $data[$j][5] &&
            $data[$i][6] + 240 > $data[$j][5] &&
            $data[$i][3] eq $data[$j][2] &&
            $data[$i][2] ne $data[$j][3]) {
            print “$data[$i][0]\t$data[$j][1]\t$data[$i][2]\t”;
            print “$data[$j][3]\t$data[$i][4]\t$data[$i][5]\t”;
            print “$data[$j][6]\t$data[$i][3]\n”;
        }
    }
}
```

New algorithm

```
open(FILE, "air.txt");
$numobs= 0;
$line = <FILE>;
while($line = <FILE>) {
    my @data_line = split(/\t|\n|\r/, $line);
    push(@data, [@data_line] );
    $numobs++;
}
close(FILE);

%originHash = ();
for ($i = 0; $i < $numobs; $i++) {
    $originHash{$data[$i][2]} = $originHash{$data[$i][2]} . " " . $i;
}
for ($i = 0; $i < $numobs; $i++) {
    $str = $originHash{$data[$i][3]};
    if ($str ne "") {
        @vals = split(" ", $str);
        for ($k = 0; $k <= $#vals; $k++) {
            $j = $vals[$k];
            if ($data[$i][6] + 45 < $data[$j][5] &&
                $data[$i][6] + 240 > $data[$j][5] &&
                $data[$i][2] ne $data[$j][3]) {
                print “$data[$i][0]\t$data[$j][1]\t$data[$i][2]\t”;
                print “$data[$j][3]\t$data[$i][4]\t$data[$i][5]\t”;
                print “$data[$j][6]\t$data[$i][3]\n”;
            }
        }
    }
}
```

Runtime

- New algorithm runs in 9 seconds with a file of 9837 flights and 52 airport codes
- Old algorithm runs in 5 minutes and 32 seconds
- Differences becomes much worse as input file and number of airport codes grows
  - For example, if the number of flights and airport codes increases by a factor of 10, then the new algorithm will run in ~90 seconds, while the old algorithm will run in ~500 minutes

Web crawler

```
#!/usr/bin/perl
$start = 1000;
$end   = 86000;
for ( $i = $start; $i <= $end; $i++ ) {
 $folder = int($i / 1000);
 $url= "http://www.cricketarchive.com/Archive/Scorecards/$folder/$i.html";
 print "$folder\t$i\t$url\n";
 `mkdir -p $folder`;
 `wget -q '$url' --output-document=./$folder/$i.html`;
 sleep 1;
}
```

NOTE: Type “man wget” at command-line of UNIX prompt to learn more about how to download webpages programmatically.

Marylebone Cricket Club v Cambridge University in 1854 - Windows Internet Explorer

http://www.cricketarchive.com/Archive/Scorecards/1/1000.html

Live Search

Marylebone Cricket Club v Cambridge University in 1854

Page Tools

f705

Marylebone Cricket Club v Cambridge University

University Match 1854

Venue	Lord's Cricket Ground, St John's Wood on 22nd, 23rd June 1854 (2-day match)
Balls per over	4
Toss	Cambridge University won the toss and decided to bat
Result	Marylebone Cricket Club won by 9 wickets
Close of play day 1	Cambridge University (2) 11/1 (Knight 6*)

Cambridge University first innings		Runs	Balls	Mins	4s	6s
EW Blore	b Nixon	14				
PH Knight	b Nixon	4				
WM Leake	b Nixon	1				
J McCormick	b Nixon	5				
*AR Ward	lbw b Dean	45				

Cambridge University v England XI in 1876 - Windows Internet Explorer

http://www.cricketarchive.com/Archive/Scorecards/2/2000.html

Live Search

Cambridge University v England XI in 1876

Page Tools

gland won

bbb of England (570-7d) v West Indies (146) (141) - En

f1696

Cambridge University v England XI

University Match 1876

Venue	FP Fenner's Ground, Cambridge on 8th, 9th, 10th May 1876 (3-day match)
Balls per over	4
Toss	Cambridge University won the toss and decided to bat
Result	Match drawn
Umpires	J Potter, J Robinson
Close of play day 1	England XI (1) 28/0 (Gilbert 9*, WG Grace ?*)
Close of play day 2	England XI (1) 359/8 (Gilbert 183*, Hoare 16*)

Cambridge University first innings		Runs	Balls	Mins	4s	6s
*FFJ Greenfield	c Hoare b McIntyre	6				
AP Lucas	c Gilbert b WG Grace	105				
DQ Steel	b Henderson	17				
E Lyttelton	b GF Grace	23				
+A Lyttelton	c McIntyre b WG Grace	78				
WS Patterson	b WG Grace	12				
HT Allsopp	c Walker b WG Grace	16				
SC Newton	b WG Grace	9				
EW Stocks	b WG Grace	1				

Web crawler with cookies

```
#!/usr/bin/perl

$cookies = "/bbkinghome/noto/.mozilla/firefox/a5gqk1zd.default/cookies.txt";
$home    = "/bbkinghome/noto/consoles";
$date    = "20070115";

$filename = $ARGV[0];
open(FILE, $filename);
$j = 0;
while($line = <FILE>) {
    $item = $line;
    $item =~ s/\t|\r|\n//g;
    print STDERR "doing item=$item \t j=$j ...\n";

    $url1 = "http://offer.ebay.com/ws/eBayISAPI.dll?ViewItem&item=$item";
    `wget -q --load-cookies $cookies  --output-document=$home/${date}_${j}.html '$url1'`;
    #http://offer.ebay.com/ws/eBayISAPI.dll?ViewBids&item=200029922634

    $url2 = "http://offer.ebay.com/ws/eBayISAPI.dll?ViewBids&item=$item";
    `wget -q --load-cookies $cookies --output-document=$home/${date}_${j}_bids.html '$url2'`;

    $j++;
}
close(FILE);
```

Chickenfoot

Chickenfoot - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://groups.csail.mit.edu/uid/chickenfoot/

Customize Links | Free Hotmail | Windows Marketplace | Windows Media | Windows

Chickenfoot Script Editor

Untitled

Output | Patterns | Actions | Triggers

Output:

Chickenfoot for Firefox: Rewrite the Web

User Interface Design Group

Chickenfoot

- Overview
- Install
- Quick Start Guide
- Commands
- Libraries
- Examples
- Scripts Wiki
- Publications
- FAQ
- Blog
- Developers

Overview

Chickenfoot is a Firefox extension that puts a programming environment in the browser's sidebar so you can write scripts to manipulate web pages and automate web browsing. In Chickenfoot, scripts are written in a superset of Javascript that includes special functions specific to web tasks.

To participate in the growing Chickenfoot community, you can:

- Join the mailing lists:

  chickenfoot-announce@csail.mit.edu receives low-traffic announcements of new releases

  chickenfoot-users@csail.mit.edu allows Chickenfoot users to talk to each other and exchange ideas

- Submit your scripts and tricks to the Scripts Wiki

- Send bug reports and feedback to chickenfoot-developers@csail.mit.edu

Release Notes (from Chickenfeed, The Official Chickenfoot Blog)

Chickenfoot 0.9.20070430 released — April 30th, 2007 01:01 AM

Chickenfoot now supports hybrid programs consisting of both Javascript and keyword commands. Text recognized as keyword

Done

Geostat Center: County Business Patterns: County Data 1977 - 1997 - Windows Internet Explorer

http://fisher.lib.virginia.edu/collections/stats/cbp/county.html

File Edit View Favorites Tools Help

Geostat Center: County Business Patterns: County D...

UNIVERSITY of VIRGINIA LIBRARY

Geospatial & Statistical Data Center

GeoStat Home | General Info | Collections | Services | Reference Resources | Browse/Search | Questions? | VIRGO

County Business Patterns

County Data 1977 - 1997

DUE TO THE NATURE AND SIZE OF THE COUNTY BUSINESS PATTERNS DATASET, USE OF COUNTY LEVEL DATA MUST BE DONE ON A STATE-BY-STATE BASIS

Begin by choosing the state you wish to analyze at the county level

ALABAMA
ALASKA
ARIZONA
ARKANSAS
CALIFORNIA
COLORADO
CONNECTICUT
DELAWARE
DISTRICT OF COLUMBIA
FLORIDA

Submit Query | Reset

Digital Scholarship Services
University of Virginia Library • PO Box 400129
Charlottesville VA 22904-4129
phone: 434.243.8800 • fax: 434.924.1431

Geostat Home • UVa Library Home
Search the Library Web • UVa Home
Maintained by: @virginia.edu
Last Modified: Monday, November 17, 2003

Internet 100%

Geostat Center: County Business Patterns - Windows Internet Explorer

http://fisher.lib.virginia.edu/cgi-local/cbpbin/county.cgi

File Edit View Favorites Tools Help

Geostat Center: County Business Patterns

Alabama

Tallapoosa County
Tuscaloosa County
Walker County
Washington County
Wilcox County
Winston County
Statewide

INDUSTRY SELECTION

Limit record selection by choosing from the list below.

Manufacturing Division
Transportation, Communications, and Utilities Division
Wholesale Trade Division
Retail Trade Division
Finance, Insurance and Real Estate Division
Services Division
Nonclassifiable Establishments Division

SIC 81 -- Legal services
SIC 82 -- Educational services
SIC 83 -- Social services
SIC 84 -- Museums, botanical, zoological gardens
SIC 86 -- Membership organizations
SIC 87 -- Engineering & management services
SIC 89 -- Services, n.e.c.

Variable Selection

Choose your variables from the list below.

Number of Employees (Week including March 12)
Payroll() 1st Quarter
Payroll() Annual
Total Number of Establishments

Done Internet 100%

Chickenfoot, con’t

```
go("http://fisher.lib.virginia.edu/collections/stats/cbp/county.html");

for(var f = find("listitem"); f.hasMatch; f = f.next) {
 var state = Chickenfoot.trim(f.text);
 output("STATE: " + state);
 pick(state);
 click("1st button");
 pick("TOTAL FOR ALL INDUSTRIES");
 pick("Week including March 12");
 pick("Payroll() Annual");
 pick("Total Number of Establishments");

 for(var year = 1977; year < 1998; year++) {
  pick(year + " listitem");
 }

 pick("Prepare the Data for Downloading");
 click("1st button");
 click("data file link");
 var body = find(document.body);
 write("cbp/" + state + ".csv", body.toString());
 output("going to new page ...");
 go("http://fisher.lib.virginia.edu/collections/stats/cbp/county.html");
 output("done!");
}
```

Where to learn more …

- Chickenfoot: http://groups.csail.mit.edu/uid/chickenfoot/

- Perl:
  - ActivePerl,
  - www.perl.com
  - www.perl.org