Lecture 10

Regular Expressions
Parsing
Javascript

Prof. Steven Ludtke
N410.07, sludtke@bcm.edu

Web Scripting (JavaScript)

Scripting, Server vs. Client

- Serverside scripting depends on the webserver you use
    - Many choices
    - May put load on server
- Clientside (with server support)
    - Java - often available, but many issues
    - Flash - No i-devices, somewhat proprietary
        - HTML5 ?
    - Javascript - built in to virtually all browsers (even tablets/phones)
        - AJAX - Asynchronous Javascript And XML

Simple Python Webserver

```
# This will serve files from the current directory
# we use port 8080 because port 80 is restricted

from BaseHTTPServer import *
from SimpleHTTPServer import *
httpd=HTTPServer(("",8080),SimpleHTTPRequestHandler)
httpd.serve_forever()
```

Simple HTML Document

```
<HTML><HEAD><TITLE>Hi there</TITLE></HEAD>
<BODY>
<h3>Here is a title</h3>
And some text
<p>
<input type="button" value="Push Me">
</p>
</body>
</HTML>
```

Javascript - Button

```
<HTML><HEAD><TITLE>Hi there</TITLE></HEAD>

<BODY>

<h3>Here is a title</h3>

And some text

<p>

<input type="button" value="Push Me"
onclick="alert('You pushed me too far')">

</p>

</body>

</html>
```

Javascript - mouseover

```
<HTML><HEAD><TITLE>Hi there</TITLE></HEAD>
<BODY>
<h3>Here is a title</h3>
And some text
<p>
<a href="index3.html" onmouseover="window.document.bgColor='red'">Red</a>
<a href="index3.html" onmouseover="window.document.bgColor='green'">Green</a>
<a href="index3.html" onmouseover="window.document.bgColor='blue'">Blue</a>
<a href="index3.html" onmouseover="window.document.bgColor='white'">White</a>
</p>
</body>
</HTML>
```

Javascript - Statements

- `var name[=value],name[=value]`
- `function f(x,y) statement`
- `if (expression) statement; else statement;`
- `do statement while (expression)`
- `while (expression) statement`
- `for ( var in array ) statement`
- `for (init; update; test) statement`

-
  ```
  switch (expr) {
    case const:
        statements
        break
    default:
        statements
  }
  ```

Javascript - Events

- onclick
- onfocus, onblur
- onmousedown, up, move, over,out
- onkeydown, up, press
- onreset
- onsubmit
- onload, unload

Javascript Calculator

```
<HTML><HEAD><TITLE>Hi there</TITLE></HEAD>
<BODY>
<h3>Calculator</h3>
<form name=calc onsubmit=compute()>
<input type=text name=data></input>
</form>
<script>
document.calc.data.value=window.location.search.split("=")[1]
function compute() {
document.calc.data.value=eval(document.calc.data.value);
}
</script>
</body>
```

Javascript - Calculator #2

```
<HTML><HEAD><TITLE>Hi there</TITLE></HEAD>
<BODY>
<h3>Calculator</h3>
<form name=calc onsubmit=compute()>
<input type=text name=data value="0"></input>
<table><tr>
<td><input type="button" value="7" onclick="num('7')"></td>
<td><input type="button" value="8" onclick="num('8')"></td>
<td><input type="button" value="9" onclick="num('9')"></td>
<td><input type="button" value="X" onclick="fn('*')"></td></tr><tr>
<td><input type="button" value="4" onclick="num('4')"></td>
<td><input type="button" value="5" onclick="num('5')"></td>
<td><input type="button" value="6" onclick="num('6')"></td>
<td><input type="button" value="-" onclick="fn('-')"></td></tr><tr>
<td><input type="button" value="1" onclick="num('1')"></td>
<td><input type="button" value="2" onclick="num('2')"></td>
<td><input type="button" value="3" onclick="num('3')"></td>
<td><input type="button" value="+" onclick="fn('+')"></td></tr><tr>
<td colspan=3><input type="button" value="0" onclick="num('0')"></td>
<td><input type="button" value="=" onclick="eql()"></td>
</tr> </table> </form>
```

Javascript - Calculator #2

```
<script>
xpr=""
rst=1
function num(val) {
        xpr+=val
        if (rst) {
                rst=0
                document.calc.data.value=""
        }
        document.calc.data.value+=val
}

function fn(val) {
        xpr+=val
        rst=1
}

function eql() {
        document.calc.data.value=eval(xpr)
        xpr=""
        rst=1
}
</script>
</body>
</html>
```

Regular Expressions

e-coli

- Find possible coding proteins from an e-coli plasmid
- Shine-Dalgarno consensus sequence (AGGAGG)
- Start (within 3-10 residues):
  - 83% ATG (3542/4284)
  - 14% GTG (612)
  - 3% TTG (103)
- Stop: TGA, TAA, TAG

in-class mini-lab

- Write a program to extract potential protein coding regions from the e-coli genome without using Biopython or regular expressions (just string manipulation)
- K-12equence can be downloaded from class website

With Strings

```
seq=file("/Volumes/Users/stevel/wp/lecture/2014_01_Intro_Programming/Lecture10/
ecoli.k12.txt","r").read()

def myfind(str,substr):
     r=str.find(substr)
     if r<0 : return ""
     return r

curloc=0
while True:
     sdloc=seq[curloc:].find("AGGAGG")
     if sdloc<0 : break

     start=curloc+sdloc+6
     subseq=seq[start:start+12]
     atg=myfind(subseq,"ATG")
     gtg=myfind(subseq,"GTG")
     ttg=myfind(subseq,"GTG")

     if min(atg,gtg,ttg)=="" :
          curloc=start
          continue
     start+=min(atg,gtg,ttg)

     srch=start
     while True:
          subseq=seq[srch:srch+3]
          print subseq,
          if subseq in ("TGA","TAA","TAG"): break
          srch+=3

     print ""
     curloc=srch
```

Regular Expressions

- ‘.’ - any character
- [abcd] - match any character in the list, may use ‘-’ or ‘^’
- ‘\s’ - any whitespace character [\t\n\r\f\v]
- ‘|’ - or, match either of 2 expressions
- (...) - used to group parts of an expression
- (?P<name>...) - a ‘named’ group (see groupdict)
- ‘*’ - 0 or more repetitions of the preceding element
- ‘+’ - 1 or more repetitions of the preceding element
- ‘?’ - 0 or 1 repetitions of the preceding element
- ‘*?’,’+?’,’??’ - non greedy version of *, + and ?
- {m,n} - match m-n copies of previous expression
- ‘^’ - start of the string
- ‘$’ - end of the string
- ..... there are more

e-coli

- Find possible coding proteins from an e-coli plasmid
- Shine-Dalgarno consensus sequence (AGGAGG)
- Start:
  - 83% ATG (3542/4284)
  - 14% (612) GTG
  - 3% (103) TTG
- Stop: TGA, TAA, TAG
- (AGGAGG[ACGT]{3,10})(ATG|GTG|TTG)(.*?)(TGA|TAA|TAG)

Regular Expressions

re functions:

- re.search(pattern,string) - search the entire string for pattern
- re.match(pattern,string) - check the beginning of the string only
- re.split(pattern,string) - much like string.split()
- re.findall(pattern,string) - list of all non-overlapping instances
- re.finditer(pattern,string) - Match object for each match
- re.sub(pattern,repl,string) - replace matches with repl

Regular Expressions

Match objects:

- group(n) - returns the matching part of the string in group n
- groups() - returns a tuple with all subgroups
- groupdict() - returns a dictionary of results based on <> names
- start(),end() - index of start or end of match

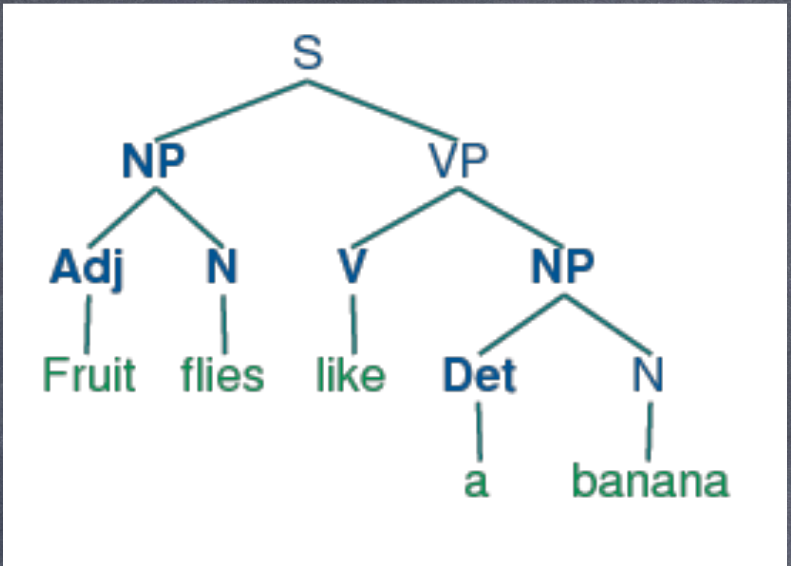
Testing Regular Expressions

- http://cthedot.de/retest/
- http://re-try.appspot.com/

Parsers

- Compilers/Interpreters
- Mathematical expressions
- Natural language

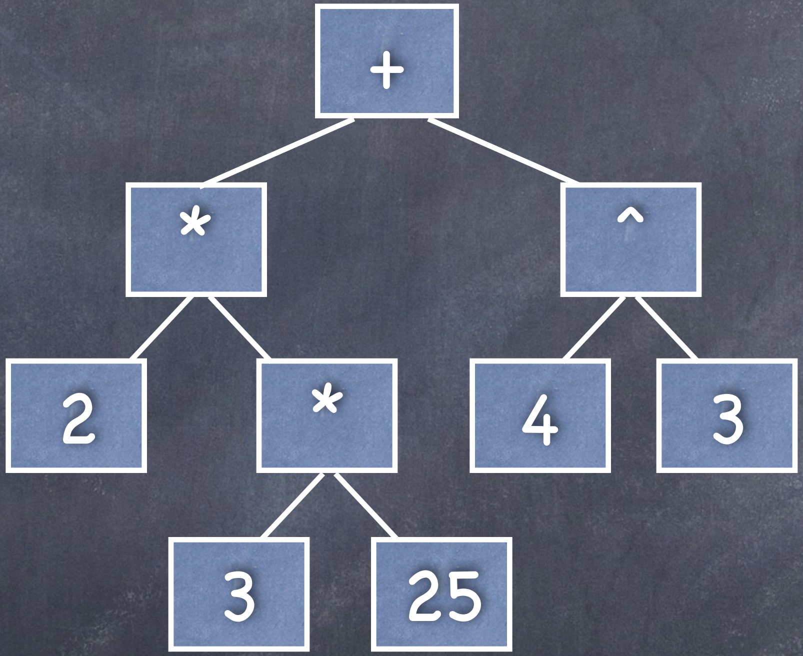
Natural Language



I run fast.
I’m going to go for a run.
The run queue on the computer is full.

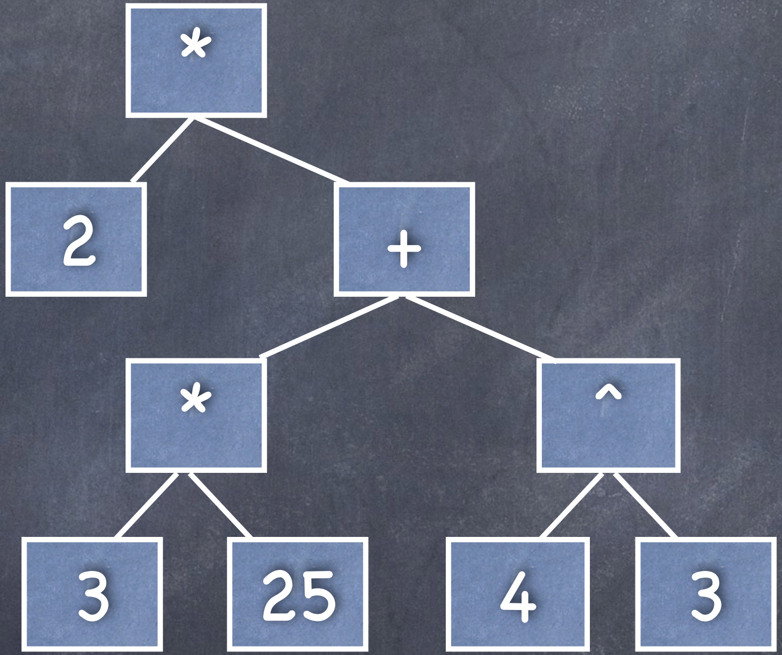
Parsing Math

2*3*25+4^3



Parsing Math

2*(3*25+4^3)



How do we generate this ?

Regular expressions ?

http://re-try.appspot.com

Parsers

- Lexical analysis
  - Search for tokens
- Parsing or Syntactic Analysis
  - Relate tokens to a ‘formal grammar’
- Evaluate Parse Tree
  - Recursion !

Parsing

- http://en.wikipedia.org/wiki/Comparison_of_parser_generators
- C/C++
  - LEX/YACC
  - Bison
- Python
  - http://wiki.python.org/moin/LanguageParsing
  - PLY (Python Lex/YACC, http://www.dabeaz.com/ply)
  - PLYPLUS (https://github.com/erezsh/plyplus)
  - http://erezsh.wordpress.com/2012/11/18/how-to-write-a-calculator-in-50-python-lines-without-eval