Lecture 13

Regular Expressions
Parsing
PyQt4

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

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

Example

- Write a program to extract potential protein coding regions from the e-coli genome

With Strings

```
seq=open("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,"TTG")

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

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

     print ""
     curloc=srch
```

Regular Expressions

- Language describing "patterns"
- Reasonably standardized across most programming languages
- Often available in applications, eg - search dialogs
- Very useful in bioinformatics, tight integration with PERL one of the reasons popular in that community
- Python is largely PERL compatible with a few extensions
- *import re*

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

Testing Regular Expressions

- http://cthedot.de/retest/
- http://re-try.appspot.com/ (doesn’t handle space?)

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

With Strings

```
seq=open("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,end="")
          if subseq in ("TGA","TAA","TAG"): break
          srch+=3

     print ""
     curloc=srch
```

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

Equivalent with Regex

```
import re
seq=open("ecoli.k12.txt","r").read()

pat="(AGGAGG)(.{3,10})(ATG|TTG|GTG)(([CATG]..)+?)(TGA|TAA|TAG)"

matches=re.findall(pat,seq)

for match in matches: print(match[2],match[3],match[-1])
```

Parsers

- Compilers/Interpreters
- Mathematical expressions
- Natural language

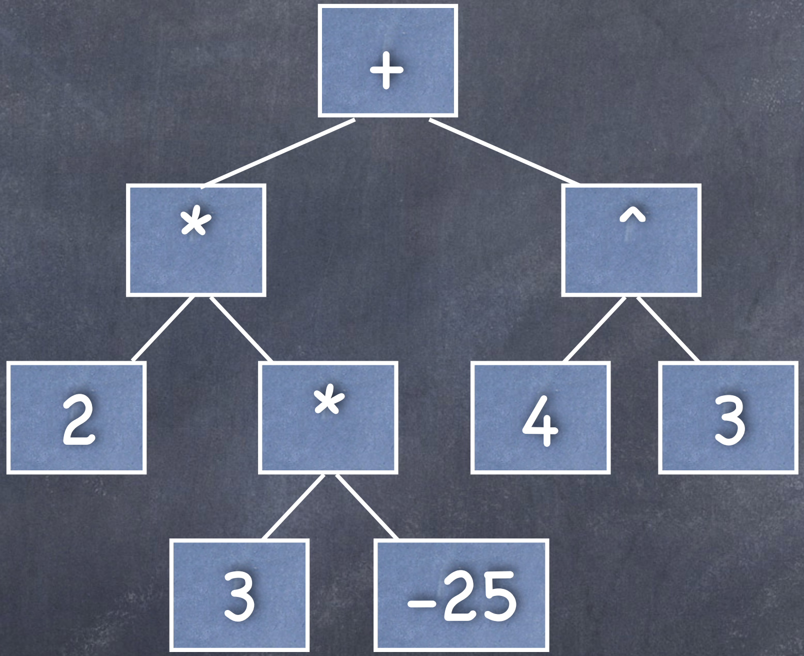
Parsing Math

2*3*-25+4^3

```
(-?[.0-9]*)([*/+^-])?
```

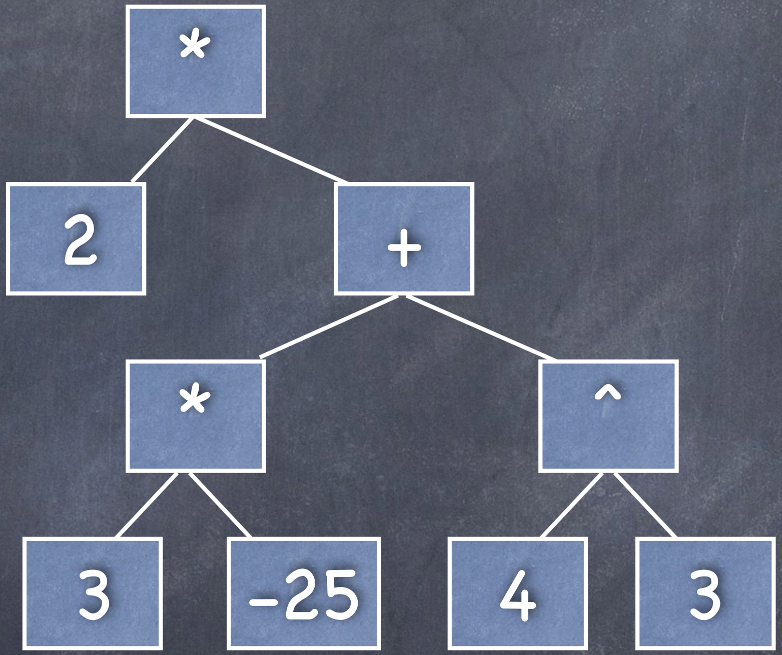
Parsing Math

2*3*-25+4^3



Parsing Math

2*(3*-25+4^3)

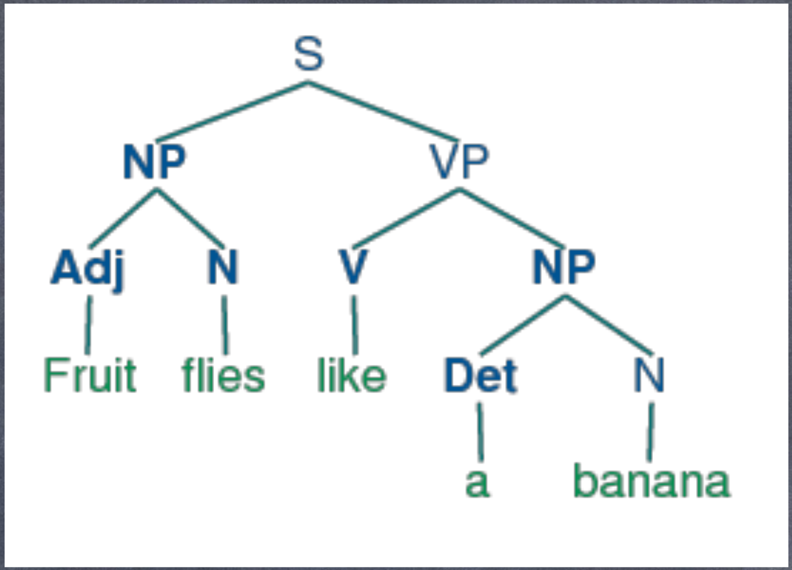


How do we generate this ?

Regular expressions ?

http://re-try.appspot.com

Natural Language



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

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

Back to GUI Programming

Qt 4.x

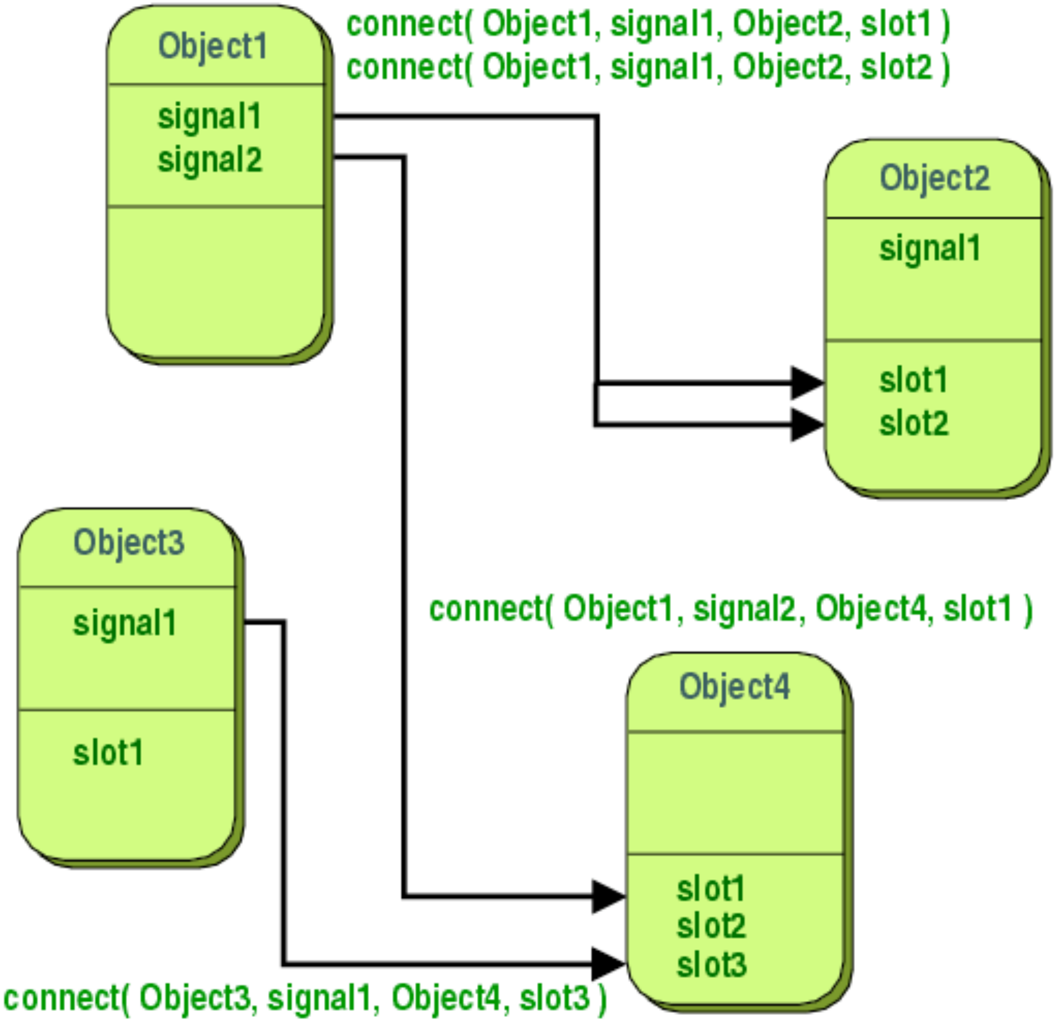
- Qt:
  - http://www.qt.io/
  - Docs: http://doc.qt.io/qt-4.8/index.html
  - Ref: http://doc.qt.io/qt-4.8/classes.html
- PyQt:
  - http://www.riverbankcomputing.co.uk/software/pyqt/intro
  - docs: http://www.riverbankcomputing.co.uk/static/Docs/PyQt4/html/classes.html
- Note that Qt5 has been out for some time, but Qt4 is still more widely used.

Graphical Layout Design

- Qt Creator - GUI design (separate install)
- uic - Build C++ code from designs
- pyuic4 - Build python code from designs

- Gallery: http://doc.qt.io/qt-4.8/gallery-macintosh.html

Signals and Slots



connect(Object1, signal1, Object2, slot1)
connect(Object1, signal1, Object2, slot2)

connect(Object1, signal2, Object4, slot1)

connect(Object3, signal1, Object4, slot3)

Simple Qt4 Application

```
from PyQt4 import QtCore, QtGui

# This is a class representing the main window for the application
class MyGuiWindow(QtGui.QWidget):
  def __init__(self,parent=None):
    QtGui.QWidget.__init__(self,parent)
    # setup widgets

  def respond(self,value):
    pass
    # do something

# This is the actual program.
# Create an Application object, set up widgets, and exec()
app = QtGui.QApplication([])
window = MyGuiWindow()
window.show()
app.exec()
```

Button

- Public Slots
    - void animateClick (int msec = 100)
    - void click ()
    - void setChecked (bool)
    - void setIconSize (const QSize & size)
    - void toggle ()

- Signals
    - void clicked (bool checked = false)
    - void pressed ()
    - void released ()
    - void toggled (bool checked)

Simple Qt4 Application

```
from PyQt4 import QtCore, QtGui

class MyGuiWindow(QtGui.QWidget):
    def __init__(self,parent=None):
        QtGui.QWidget.__init__(self,parent)

        # organizes the widgets into a grid
        self.gbl = QtGui.QGridLayout(self)

        # create a PushButton and add it to the window
        self.but = QtGui.QPushButton("Push Me")
        self.gbl.addWidget(self.but,0,0)

        # connect the 'clicked' signal to the respond() method
        self.but.clicked.connect(self.respond)

    def respond(self,value):
        QtGui.QMessageBox.information(None,"Ouch","That hurt!  Why did you do that?")


app = QtGui.QApplication([])
window = MyGuiWindow()
window.show()
app.exec()
```

CLASS PROJECT PRESENTATIONS

- Monday, Feb 29
- 9 AM (usual class time & location)
- We have the room until 11:30, but shouldn't need it
- You will have 10 minutes total:
    - Set up your presentation (1 minute) - TEST LAPTOP BEFORE FEB 29!!!
    - Give your talk (7 minutes)
        - What does your software do, and why did you write it
        - Inputs and outputs
        - Demonstration
    - Questions (2 minutes)
- 1/3 of your grade will be for the presentation, and 2/3 for the program itself. Combined this is 1/2 of your final grade in the class.
- The program MUST WORK to get a good grade. Better to turn in something that doesn't do everything you wanted, but works, than something broken

CLASS PROJECTS

- Must do something useful in some specific context
- Not be trivial
- If you have past programming experience I will expect more

- **Please follow these instructions exactly:**
- Your class project MUST be submitted by 11:59 PM on Sat, Feb 27. No revisions will be accepted after this time. You can use Sunday to prepare your oral presentation
- Your submission should consist of:
- one or more .py files (should have sufficient comments to figure out how they work)
- any necessary additional files to demonstrate that the program works
- A PDF file with a brief description of your program, what inputs the program takes, what outputs the program produces, and what it is supposed to do.
- The final item in the PDF should be a command-line to use in running the program, and any necessary instructions to demonstrate that it works.
- Combine all files into a .zip file named: Familyname_Givenname_project_2016.zip
- Email sludtke@bcm.edu with the subject "Class project submission", and attach the .zip file