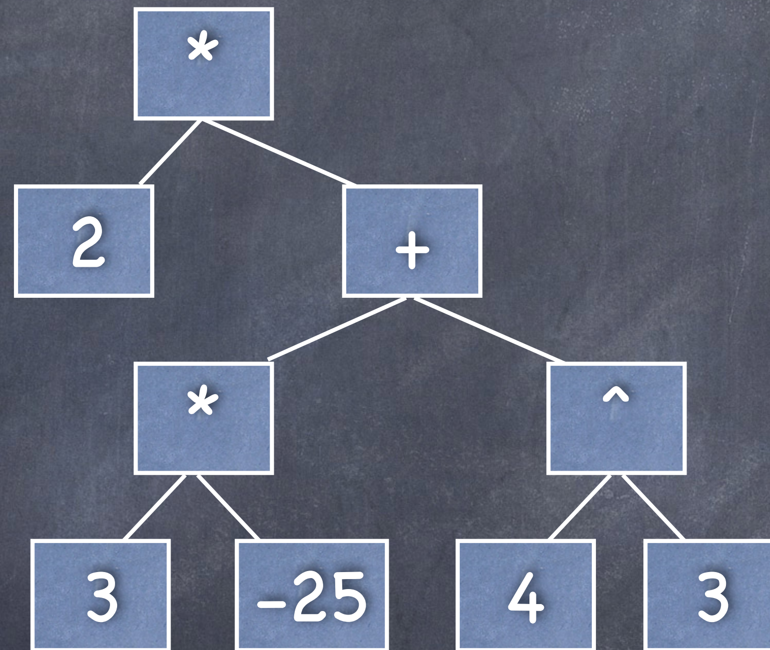
Lecture 14

Recursion
More on Web Applications

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

Parsing Math

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

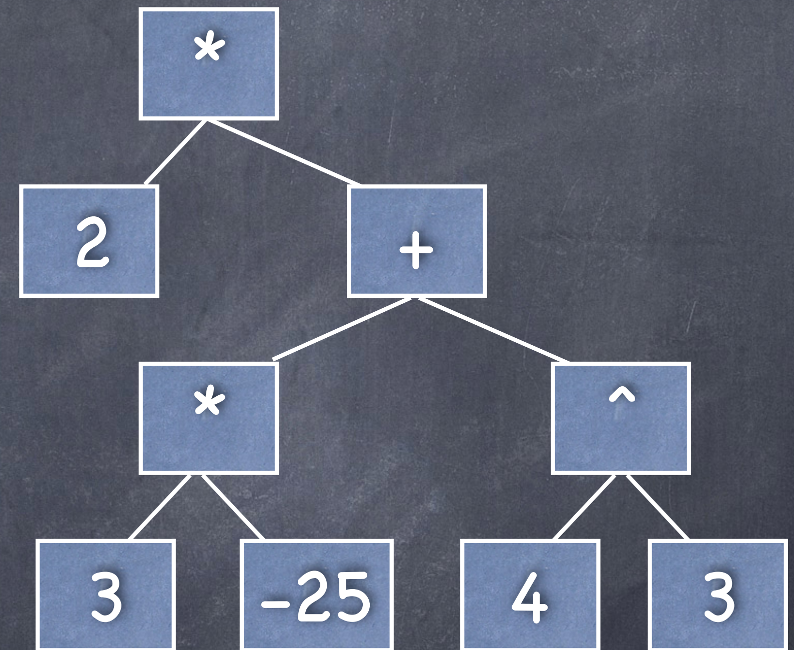


How do we execute this ?

Parsing Math

```
if isnum(root.val):
  return root.val
else:
  if isnum(root.child[0].val):
    v0=root.child[0].val
  else:
    v0=...
  if isnum(root.child[1].val):
    v1=root.child[1].val
  else:
    v1=...
  return eval(str(v0)+root.val+str(v1))
```

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



How do we execute this ?

Recursion

- A function that calls itself

```
def factorial(x):
        if x==1 : return 1
        return x*factorial(x-1)
```

‘Scrabble’ Problem

- What words could you make given these letters: PGAORRM ?

‘Scrabble’ Problem

- What words could you make given tiles containing PGAORRM
- 68 of them
- program armor gompa gramp groma maror morra pargo gamp gapo gora gorm gorp gram marg mora ogam orra parr pram prao proa prog prom ramp roam roar roma romp ago amp apo arm gam gap gar goa gor mag map mar moa mog mop mor oar ora pam par poa pom pro rag ram rap rom ag am ar go ma mo om op or pa po a

‘Scrabble’ Problem

- You have 7 random letters. What real words can you make from them ?
- How many ‘words’ could we make ?
- 7*6*5*4*3*2*1 + 7*6*5*4*3*2 + ... = 7! + 7!/1! + 7!/2! + 7!/3! + ... = 13699

‘Scrabble’ Problem

- You have 7 random letters. What real words can you make from them ?
- How many letter combinations could we make ?
- 7*6*5*4*3*2*1 + 7*6*5*4*3*2 + ... = 7! + 7!/1! + 7!/2! + 7!/3! + ... = 13699
- Different approaches:
  - Make all possible words, check to see if each is in the dictionary
    - Linear
    - Recursive
  - Check each word in the dictionary to see if it can be made from the letters in the list

Design #1

- ask for letters
- Read list of words
- Nested loop to make each possible word from letters
  - check to see if word is real
  - if so, add it to the list
- sort the list and print results

Design #2

- ask for letters
- Read list of words
- Recursion to make each possible word from letters
  - check to see if word is real
  - if so, add it to the list
- sort the list and print results

Design #3

- ask for letters
- Read list of words
- Loop over list of words
  - make a list of available letters
  - Loop over the letters in word
    - see if each letter is in the list, if so, remove
  - if we found all of the letters print the word

Results

- Nested loop : 93.8 sec
- Nested loop (set): 0.44 sec
- Recursive : 0.38 sec
- Check all words: 0.92 sec

Calling External Programs

- os.system(command)
  - simple to use, no i/o, blocks until complete
- os.popen(command,r/w)
  - one way console communications
- os.spawn*(mode,command)
  - can launch jobs in background
- os.exec*(command,…)
  - Replaces current process with new command. Exits python immediately
- subprocess module
  - preferred alternative to all of the above, but takes (slightly) more code to use

Calling External Programs

- Program interacts via console
  - Use subprocess module
- Program has a GUI
  - No good cross-platform way to automate, unless the program provides some built-in scripting capabilities
- Program takes command-line arguments, reads/writes files
  - Use os.system or subprocess module

subprocess

- subprocess.run([args],input=None, stdout=None, stderr=None, shell=False, timeout=None, check=False)
  - Did not exist before Python 3.5!
  - Flexible enough many, but not all use-cases
  - input passed to stdin if specified
  - Set stdout and/or stderr to PIPE to capture output
- subprocess.Popen(args, bufsize=-1, executable=None, stdin=None, stdout=None, stderr=None, preexec_fn=None, close_fds=True, shell=False, cwd=None, env=None, universal_newlines=False, startupinfo=None, creationflags=0, restore_signals=True, start_new_session=False, pass_fds=())
  - more flexibility = more complexity
  - subprocess.communicate() to send/receive data
- Possible to interact with programs expecting truly interactive console sessions, but may be very complicated to make it work without deadlocks.

Simple Python Webserver

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

from http.server import *

httpd = HTTPServer(("",8080),SimpleHTTPRequestHandler)
httpd.serve_forever()
```

Simple Python Webserver

```
from http.server import *
import os
from http import HTTPStatus

class myHandler(BaseHTTPRequestHandler):
    def do_GET(self):
        ret=b"It's OK\nI'm here at:"+bytearray(self.path,"utf-8")
        self.send_response(HTTPStatus.OK)
        self.send_header("Content-type", "text/plain")
        self.send_header("Content-Length", str(len(ret)))
        self.end_headers()
        self.wfile.write(ret)
        return

httpd = HTTPServer(("",8080),myHandler)
httpd.serve_forever()
```

Pythonic Web Application Servers

- Django
  - Mature, complete, many developers, stable. Deep learning curve.
- CherryPy
  - Old and stable, but minimalistic.
- Flask
  - A "microframework", easy to use, but relies on other packages for much functionality
- Pyramid
  - A "general open source Python framework"
- Grok
  - Python-centric, emphasis on agile development and as good for beginners
- TurboGears
  - "The web framework that scales with you!"
- ... a dozen more

Web2Py

- A single installable package (Mac/Windows) with everything you need
- Zero configuration
- Web-based management and editing
- Web-based database model editor
- RDB abstraction
- Integration with Google App Engine
- Very easy to use for new programmers

- However, Python2, not Python3, and they aren't anxious to switch