# A Very Simple LaTeX $2_\varepsilon$ Template

Vitaly Surazhsky
Department of Computer Science
Technion—Israel Institute of Technology
Technion City, Haifa 32000, Israel

Yossi Gil
Department of Computer Science
Technion—Israel Institute of Technology
Technion City, Haifa 32000, Israel

May 18, 2012

## 1 python

### 1.1 Testing

```
import time

examples = """TWO + TWO == FOURi
A**2 + B**2 == C**2""".splitlines()

def test():
  t0 = time.clock()
  for example in examples:
    print; print 13*' ',example
    print '%6.4f sec:   %s ' % timedcall(solve, example)
  print '%6.4f tot.' % (time.clock()-t0)

test()
```

### 1.2 Profiling

```
#Terminal
$ python -m cProfile file.py

#From within python
import cProfile
cProfile.run('function()')
```

## 1.3 string

### 1.3.1 String Substitution

```
s = "some bold"
print "<b>%s</b> text" % s
#--> <b>some bold</b> text
s2 = "some italic"
print "<b>%s</b> and <i>%s</i> text" % (s,s2)
#--> <b>some bold</b> and <i>some italic</i> text
print "I'm %(nickname)s. My real name is %(name)s, but my friends call me %(nickname)s." % {'name':'Mike','nickname':'
#--> I'm Goose. My real name is Mike, but my friends call me Goose.
```

### 1.3.2 split

```
#splitting by whitespace
"python is kind of fun".split()
```

## 1.4 hash

Implements a hash function, that is irreversible, gives a unique output and changing the input just a litte, gives a totally different output.

* **crc32** - Designed for checksums. It's fast, but its security properties are not very good - it's very easy to find a *collision*, when two things hash to the same value.
* **md5** - Used to be used because it was pretty fast and pretty secured. But it's not secure any longer - it's easy to find collisions. It's found on every system.
* **sha1** - Not as fast, but fairly secured. Second most widely used hash after md5.
* **sha256** - If you are really worried about securing your data, but it's the slowest.
* **bcrypt** - Besides incorporating a salt to protect against rainbow table attacks, bcrypt is an adaptive hash: over time it can be made slower and slower so it remains resistant to specific brute-force search attacks against the hash and the salt.

### 1.4.1 get

Does not generate a *KeyError* when referencing a key that does not exist in the hash, unlike using the sqare brackets.

```
h = {'ruby':'rocks'}
h.get('python')
#--> <nothing>
h['python']
#--> KeyError: 'python'
```

### 1.4.2 defaultdict

```
from collections import defaultdict
cache = defaultdict(int)
for c in 'abcd':
  cache[c] += 1
print cache #--> defaultdict(<type 'int'>, {'a': 1, 'c': 1, 'b': 1, 'd': 1})
```

### 1.4.3 merge

Merge two or more dictionaries

```
a = {'a':1}; b = {'b':2}
print dict(a.items()+b.items()) #--> {'a': 1, 'b': 2}
```

## 1.5 list

### 1.5.1 map

```
def mysquare(x): return x*x
print map(mysquare,[1,2,3]) #-->[1, 4, 9]

#lambda below is called an anonymous function
print map(lambda(x):x*x,[1,2,3]) #-->[1, 4, 9]

#list comprehension
print [x*x for x in [1,2,3]] #-->[1, 4, 9]
```

## 1.6 itertools

### 1.6.1 permutations

```
import itertools

#In how many ways can five numbers be ordered?
orderings = list(itertools.permutations([1,2,3,4,5]))
print len(orderings)
#--> 120

#In how many ways can ten numbers be ordered in groups of three?
orderings = list(itertools.permutations('1234567890',3))
print len(orderings)
#--> 720
```

## 1.7 random

### 1.7.1 choice

```
import string
import random
import doctest
def make_salt():
  """5 random characters
  >>> len(make_salt())
  5
  """
  return "".join(random.choice(string.letters) for _ in xrange(5))
print doctest.testmod()
```

## 1.8 range vs xrange

range creates a list in memory, but xrange is a generator, so it evaluates lazily.

## 1.9 lambda

```
mystery = lambda(x):x+2
print mystery(3) #-->5
```

## 1.10 Something

### 1.10.1 __name__

```
def debug_fn(f):
  """Return a modified function that first prints out
  function name and arguments
  then calls the function.""" # this is the doc string
  def _f(*args):
      """ Here's the bit that prints out the name and args"""
      print "Called %s(%s)"%(f.__name__, ', '.join(map(repr, args)))
      return f(*args)
  return _f

print debug_fn.__name__
# --> debug_fn
```

### 1.10.2 __doc__

```
print debug_fn.__doc__
# --> Return a modified function that first prints out
# -->   function name and arguments
# -->   then calls the function.
```

## 1.11 classes

```
from collections import namedtuple
Link = namedtuple('Link',['id','url'])
print Link(1,'google.com') #--> Link(id=1, url='google.com')
```

## 1.12 Structured Query Language - SQL

Invented in the 1970s.

```
import sqlite3
from collections import namedtuple
Link = namedtuple('Link',['id','url'])

db = sqlite3.connect(':memory:')
db.execute('create table links (id integer, url text)')
db.execute('insert into links values (?,?)',Link(1,'google.com'))
cursor = db.execute('select url from links')
for link_tuple in cursor:
  print link_tuple #--> (u'google.com',)
cursor = db.execute('select * from links')
print Link(*cursor.fetchone()).url #--> google.com
```

### 1.12.1 index

A *sequential scan* doesn't work fine when you have a million rows to scan. Indexes increase the speed of database reads, but probably decrease the speed of database inserts, since the indexes have to be updated.

```
explain analyze select name from users where id = 123;
create index user_id on users(id);
drop index user_id;
```

**hashtable**, not sorted, lookup in constant time

**tree**, sorted, lookup in the log n order

## 1.13 ACID

**Atomicity.** All parts of a *transaction* succeed or fail together.

**Consistency.** The database will always be consistent. The database will move from one valid transaction to the next. *Replication lag* is an example of the loss of consistency.

**Isolation.** No transaction can interfere with another's. Sometimes accomplished by *locking*.

**Durability.** Once the transaction is comitted, it won't be lost.

## 1.14 Generator Expressions

* less indentation, compared to nested for-loops
* stop early, compared to a list comprehension that has to do all the work
* easy to edit, easy to move around constraints without having to worry about getting the indentation right

```
def sq(x): print 'sq called', x; return x*x
g = (sq(x) for x in range(10) if x%2 == 0)
next(g)
#--> sq called 0
next(g)
#--> sq called 2
next(g)
#--> sq called 4
next(g)
#--> sq called 6
next(g)
#--> sq called 8
next(g)
#..> ...
#--> StopIteration

#To not bother dealing with the StopIteration, use a for-loop
for x2 in (sq(x) for x in range(10) if x%2 == 0): pass
#--> sq called 0
#--> sq called 2
#--> sq called 4
#--> sq called 6
#--> sq called 8

print list((sq(x) for x in range(10) if x%2 == 0))
#--> sq called 0
#--> sq called 2
#--> sq called 4
#--> sq called 6
#--> sq called 8
#--> [0, 4, 16, 36, 64]
```

## 1.15 Generator Functions

Allows us to deal with infinite sequences.

```
def ints(start,end=None):
  i = start
  while i <= end or end is None:
    yield i
    i += 1
```

```
L = ints(0,10**6)
print L
#--> <generator object ints at 0x7fe4f0613960>

print next(L)
#--> 0
```

Print only odd numbers

```
def odds_only(ns):
  for n in ns:
    if n%2==1: yield n
print [x for x in odds_only([1,2,3,4,5])] #--> [1, 3, 5]

#list comprehension with a guard or a predicate
print [x for x in [1,2,3,4,5] if x%2==1] #--> [1, 3, 5]
```

## 1.16 Decorator Notation

### 1.16.1 Function Mapping - An expressiveness tool

Extend a binary function to a function that can take any number of arguments.

```
def n_ary(f):
  """Given binary function f(x, y), return an n_ary function such
  that f(x, y, z) = f(x, f(y,z)), etc. Also allow f(x) = x."""
  def n_ary_f(x, *args):
      return x if not args else f(x,n_ary_f(*args))
  return n_ary_f

def seq(x,y): return('seq',x,y)
seq = n_ary(seq)
```

This pattern is so common in python, that there's a special notation for it, called the Decorator Notation.

```
@n_ary
def seq(x,y): return ('seq',x,y)
```

But there is one problem with how we have specfied this decorator. In an interactive session, asking for help on seq, it is called *n_ary_f*. To fix this:

```
from functools import update_wrapper
def n_ary(f):
  """Given binary function f(x,y), return an n_ary function such that
  f(x,y,z) = f(x,f(y,z)), etc. Also allow f(x) = x."""
  def n_ary_f(x,*args):
    return x if not args else f(x,n_ary_f(*args))
  update_wrapper(n_ary_f,f)
  return n_ary_f
```

We have, though, introduced some repetition now. We always want to update the wrapper for every decorator.

```
from functools import update_wrapper
def decorator(d):
  "Make function d a decorator: d wraps a function fn."
  def _d(fn):
    return update_wrapper(d(fn),fn) #update wrapper for decorated function
  update_wrapper(_d,d) #update wrapper for decorator
  return _d

@decorator
def n_ary(f):
  """Given binary function f(x,y), return an n_ary function such that
  f(x,y,z) = f(x,f(y,z)), etc. Also allow f(x) = x."""
  def n_ary_f(x,*args):
    return x if not args else f(x,n_ary_f(*args))
  return n_ary_f
```

or

```
from functools import update_wrapper
def decorator(d):
  "Make function d a decorator: d wraps a function fn."
  return lambda fn: update_wrapper(d(fn),fn)
decorator = decorator(decorator)
```

### 1.16.2 Memoization - A performance tool

```
from decorators import decorator
@decorator
def memo(f):
  """Decorator that caches the return value for each call to f(args).
  Then when called again with same args, we can just look it up."""
  cache = {}
  def _f(*args):
    try:
      return cache[args]
    except KeyError:
      cache[args] = result = f(*args)
      return result
    except TypeError:
      # some element of args can't be a dict key (list,dic - mutable)
      return f(args)
  return _f
```

### 1.16.3 Count Calls - A debugging tool

```
from __future__ import division
from decorators import decorator,memo

@decorator
def countcalls(f):
  def _f(*args):
    callcounts[_f] += 1
    return f(*args)
  callcounts[_f] = 0
  return _f
callcounts = {}
```

```
prev_calls = 1
for n in range(32):
  @countcalls
  def fib(n): return 1 if n<=1 else fib(n-1) + fib(n-2)
  result,calls = fib(n),callcounts[fib]
  print '%2d %7d %8d %1.4f' % (n,result,calls,calls/prev_calls)
  prev_calls = calls
# ...
# --> 30 1346269  7049123 1.6180
# --> 31 2178309 11405740 1.6180

for n in range(32):
  @countcalls
  @memo
  def fib(n): return 1 if n<=1 else fib(n-1) + fib(n-2)
  result,calls = fib(n),callcounts[fib]
  print '%2d %7d %8d %1.4f' % (n,result,calls,calls/prev_calls)
  prev_calls = calls
# ...
# --> 30 1346269       59 1.0351
# --> 31 2178309       61 1.0339
```

Observe the ratio $1.618 = (1 + \sqrt{5})/2$, *the Golden Ratio*.

### 1.16.4 Trace - A debugging tool

```
from decorators import decorator
@decorator
def trace(f):
  indent = '  '
  def _f(*args):
    signature = '%s(%s)' % (f.__name__, ', '.join(map(repr,args)))
    print '%s--> %s' % (trace.level*indent,signature)
    trace.level += 1
    try:
      result = f(*args)
      print '%s<-- %s == %s' % ((trace.level-1)*indent,signature,result)
    finally:
      trace.level -= 1
    return result
  trace.level = 0
  return _f

@trace
def fib(n): return 1 if n<=1 else fib(n-1)+fib(n-2)
fib(3)
# --> --> fib(3)
# -->   --> fib(2)
# -->     --> fib(1)
# -->     <-- fib(1) == 1
# -->     --> fib(0)
# -->     <-- fib(0) == 1
# -->   <-- fib(2) == 2
# -->   --> fib(1)
# -->   <-- fib(1) == 1
# --> <-- fib(3) == 3
```

### 1.16.5 Disabled - A debugging tool

Another name for the identity function. If one of the debug tools are being used (let's say *trace*) and being scattered all around a file, it's easy to disable it by just:

```
def disabled(f): return f
trace = disabled
```

## 1.17 The Law of Diminishing Returns

## 1.18 for-loops

```
for x in items: print x

#python does the conversion
it = iter(items)
try:
  while True:
    x = next(it)
    print x
except StopIteration:
  pass
```

## 1.19 Substring

```
print 'reverse'[::-1]
#--> esrever, reverse a string
```

## 1.20 Benchmarking

```
import time

def timedcall(fn,*args):
  "Call function with args; return the time in seconds and result."
  t0 = time.clock()
  result = fn(*args)
  t1 = time.clock()
  return t1-t0,result

def timedcalls(n, fn, *args):
  """Call fn(*args) repeatedly: n times if n is an int, or up to
  n seconds if n is a float; return the min, avg and max time."""
  if isinstance(n,int):
    times = [timedcall(fn,*args)[0] for _ in range(n)]
  else:
    times = []
    while sum(times) < n:
      times.append(timedcall(fn,*args)[0])
  return min(times), average(times), max(times)

def average(n):
  "Return the average (arithmetic mean) of a sequence of numbers."
  return sum(n) / float(len(n))

def loop(stop):
```

```
  for _ in range(stop): pass

print timedcalls(10, loop,10**6)
#--> (0.02, 0.028, 0.04)

print timedcalls(10., loop,10**6)
#--> (0.02, 0.027, 0.04) takes 10s
```

## 1.21 Translation Table

```
import string

table = string.maketrans('ABC','123')
f = 'A+B==C'
print eval(f.translate(table))
#--> True
```

## 1.22 Future Imports

In python 2.x, you can do integer division. In python 3, integer division returns a float. If you want this kind of behaviour in python 2.x, do

```
from __future__ import division
```

## 1.23 Regular Expressions

Regular expressions describe regular languages and can be represented by a FSM. If a language is regular, then that language is also context free. The module to import in python is called re. A regular expression is written

### 1.23.1 findall

```
import re
print re.findall(r"[0-9]","1+2==3")
#--> ['1','2','3']
print re.findall(r"[0-9][0-9]","12345")
#--> ['12','34']
print re.findall(r"[0-9]+","13 from 1 in 1776")
#--> ['13', '1', '1776'] Maximal Munch. Don't stop early. go all the way
print "".join(set(re.findall(r'[A-Z]','I+I=ME')))
#--> IEM, Find all unique capital letters
```

where the $r$ actually means *raw string* instead of *regular expression*. The + and * operators are called *Kleene Operators* after Stephen C. Kleene.

### 1.23.2 search

```
import re
#Find a str where the first digit of a multi-digit number is 0
print re.search(r'\b0[0-9]','400 + 5 == 0405')
#--> <_sre.SRE_Match object at 0x7f611819f098>
```

j

| | |
|---|---|
| \b | word boundary |
| * | Kleene Operator |
| + | Kleene Operator |

### 1.23.3 split

```
import re
print re.split('([A-Z]+)','YOU == ME ** 2')
#--> ['', 'YOU', ' == ', 'ME', ' ** 2']
```

## 1.24 sub

```
import re
print re.sub(r"[0-9]+", "NUMBER", "22 + 33 = 55")
#--> NUMBER + NUMBER = NUMBER
```

j

Grammar describe context free languages, a more powerful set of strings than regular languages, and is represented by context free grammar. If a language is context-free, then that language is sometimes also regular.

r'ab' $\Rightarrow$ g $\rightarrow$ a b
r'a*' $\Rightarrow$ g $\rightarrow \epsilon$
g $\rightarrow$ a g
r'a|b' $\Rightarrow$ g $\rightarrow$ a
g $\rightarrow$ b

It's f.ex. impossible to capture balanced parenthesis with regular expressions.
A grammar is abiguous if at least 1 string in the grammar has more than 1 different parse tree.

## 1.25 Problems

### 1.25.1 Water Pouring Problem

```
import doctest

def pour(x,y,X,Y): return {((0,x+y) if x+y<=Y else (x-(Y-y),y+(Y-y))):'X->Y',
                           ((x+y,0) if x+y<=X else (x+(X-x),y-(X-x))):'Y->X'}
def empty(x,y,X,Y): return {(0,y):'empty X',(x,0):'empty Y'}
def fill(x,y,X,Y): return {(X,y):'fill X',(x,Y):'fill Y'}

def successors(x,y,X,Y):
  """Return a dict of {state:action} paris describing what can be reached
  from the (x,y) state, and how."""
  assert x<=X and y<=Y
  return dict(fill(x,y,X,Y).items() + empty(x,y,X,Y).items() + pour(x,y,X,Y).items())

def pour_problem(X,Y,goal,start=(0,0)):
  """X and Y are the capacity of the two glasses; (x,y) is current fill levels
  and represents a state. The goal is a level that can be in either glass.
  Start at the start state and follow successors until we reach the goal.
  Keep track of frontier and previously explored; fail when no frontier."""
  if goal in start: return [start]
  explored = set()
  frontier = [[start]]
  while frontier:
    path = frontier.pop(0)
    (x,y) = path[-1]
```

```
    for (state,action) in successors(x,y,X,Y).items():
      if state not in explored:
        explored.add(state)
        npath = path + [action,state]
        if goal in state: return npath
        else: frontier.append(npath)

class Test: """
>>> pour(1,1,9,4)
{(2, 0): 'Y->X', (0, 2): 'X->Y'}
>>> pour(8,3,9,4)
{(7, 4): 'X->Y', (9, 2): 'Y->X'}

>>> empty(2,3,9,4)
{(2, 0): 'empty Y', (0, 3): 'empty X'}

>>> fill(2,3,9,4)
{(9, 3): 'fill X', (2, 4): 'fill Y'}

>>> successors(2,3,9,4)
{(9, 3): 'fill X', (1, 4): 'X->Y', (2, 0): 'empty Y', (5, 0): 'Y->X', (0, 3): 'empty X', (2, 4): 'fill Y'}
"""
print doctest.testmod() #--> TestResults(failed=0, attempted=5)

print pour_problem(9,4,6)
#--> [(0, 0), 'fill X', (9, 0), 'X->Y', (5, 4), 'empty Y', (5, 0), 'X->Y',
#-->  (1, 4), 'empty Y', (1, 0), 'X->Y', (0, 1), 'fill X', (9, 1), 'X->Y', (6, 4)]
```

### 1.25.2 The Bridge - Lowest Cost Search

```
fs = frozenset

def bridge_problem(here):
  "Find the fastest (least elapsed time) path to the goal in the bridge problem."
  start = (fs(here) | fs(['light']), fs())
  return lowest_cost_search(start, bsuccessors, all_over, bcost)

def all_over(state):
  here, _ = state
  return not here or here == set('light')

def final_state(path): return path[-1]

def path_cost(path):
  "The total cost of a path (which is stored in a tuple with the final action)."
  if len(path) < 3: return 0
  _, total_cost = path[-2]
  return total_cost

def add_to_frontier(frontier,path):
  """Add path to frontier, replacing costlier path if there is one.
  (This could be done more efficiently.)"""
  old = None
  for i,p in enumerate(frontier):
    if final_state(p) == final_state(path):
      old = i
      break
```

```
  if old is not None and path_cost(frontier[old]) < path_cost(path):
    return # Old path was better; do nothing
  elif old is not None:
    del frontier[old] # Old path was worse; delete it
  # Now add the new path and re-sort
  frontier.append(path)
  frontier.sort(key=path_cost)

def lowest_cost_search(start, successors, is_goal, action_cost):
  """Return the lowest cost path, starting from start state
  and considering successors(state) => {state:action,...},
  that ends in a state for which is_goal(state) is true,
  where the cost of a path is the sum of action costs,
  which are given by action_cost(action)."""
  explored = set()
  frontier = [[start]]
  while frontier:
    path = frontier.pop(0)
    nstate = final_state(path)
    if is_goal(nstate): return path
    explored.add(nstate)
    pcost = path_cost(path)
    for (state,action) in successors(nstate).items():
      if state not in explored:
        total_cost = pcost + action_cost(action)
        npath = path + [(action,total_cost),state]
        #don't check for solution here
        add_to_frontier(frontier,npath)
  return []

def bcost(action):
  """ Returns the cost (a number) of an action in the bridge problem.
  An action is an (a,b,arrow) tuple; a and b are times; arrow is a string"""
  a, b, _ = action
  return max(a,b)

def bsuccessors(state):
  """Return a dict of {state:action} pairs. A state is a (here, there) tuple,
  where here and there are frozensets of people (indicated by their times) and/or
  the 'light."""
  here, there = state
  if 'light' in here:
    return dict(((here - fs([a,b,'light']), there | fs([a,b,'light'])), (a,b,'->'))
                for a in here if a is not 'light' for b in here if b is not 'light')
  else:
    return dict(((here | fs([a,b,'light']), there - fs([a,b,'light'])), (a,b,'<-'))
                for a in there if a is not 'light' for b in there if b is not 'light')

def test():
  assert bsuccessors((fs([1,'light']), fs([]))) == {(fs([]),fs([1,'light'])):(1, 1, '->')}
  assert bsuccessors((fs([]), fs([2,'light']))) == {(fs([2,'light']),fs([])):(2, 2, '<-')}
  here = [1,2,5,10]
  assert bridge_problem(here) == [
          (frozenset([1, 2, 'light', 10, 5]), frozenset([])),
          ((2, 1, '->'), 2),
          (frozenset([10, 5]), frozenset([1, 2, 'light'])),
          ((2, 2, '<-'), 4),
```

```
          (frozenset(['light', 10, 2, 5]), frozenset([1])),
          ((5, 10, '->'), 14),
          (frozenset([2]), frozenset([1, 10, 5, 'light'])),
          ((1, 1, '<-'), 15),
          (frozenset([1, 2, 'light']), frozenset([10, 5])),
          ((2, 1, '->'), 17),
          (frozenset([]), frozenset([1, 10, 2, 5, 'light']))]
  return 'test passes'

print test()
print bridge_problem([1,2,5,10])
#--> [(frozenset([1, 2, 'light', 10, 5]), frozenset([])), ((2, 1, '->'), 2),
#-->  (frozenset([10, 5]), frozenset([1, 2, 'light'])), ((2, 2, '<-'), 4),
#-->  (frozenset(['light', 10, 2, 5]), frozenset([1])), ((5, 10, '->'), 14),
#-->  (frozenset([2]), frozenset([1, 10, 5, 'light'])), ((1, 1, '<-'), 15),
#-->  (frozenset([1, 2, 'light']), frozenset([10, 5])), ((2, 1, '->'), 17),
#-->  (frozenset([]), frozenset([1, 10, 2, 5, 'light']))]
```

### 1.25.3 Missionaries and Cannibals - Shortest Path Search

```
def shortest_path_search(start, successors, is_goal):
  """Find the shortest path from start state to a state
  such that is_goal(state) is true."""
  if is_goal(start): return [start]
  explored = set()
  frontier = [[start]]
  while frontier:
    path = frontier.pop(0)
    s = path[-1]
    for (state,action) in successors(s).items():
      if state not in explored:
        explored.add(state)
        npath = path + [action,state]
        if is_goal(state): return npath
        else: frontier.append(npath)
  return []

def mc_problem(start=(3,3,1,0,0,0),goal=None):
  """Solve the missionaries and cannibals problem.
  State is 6 ints: (M1, C1, B1, M2, C2, B2) on the start (1) and other (2) side.
  Find a path that goes from the initial state to the goal state (which, if
  not specified, is the state with no people or boats on the start side."""
  if goal is None:
    def goal_fn(state): return state[:3] == (0,0,0)
  else:
    def goal_fn(state): return state == goal
  return shortest_path_search(start,csuccessors,goal_fn)

def csuccessors(state):
  """Find successors (including ones that result in dining) to this state.
  But a state where cannibals can dine has no successors."""
  M1, C1, B1, M2, C2, B2 = state
  ## Check for state with no successors
  if C1 > M1 > 0 or C2 > M2 > 0: return {}
  if B1 > 0: items = [(sub(state,delta), a + '->') for delta,a in deltas.items()]
  if B2 > 0: items = [(add(state,delta), '<-' + a) for delta,a in deltas.items()]
  return dict(items)
```

```
def add(X,Y): return tuple(x+y for x,y in zip(X,Y))
def sub(X,Y): return tuple(x-y for x,y in zip(X,Y))

deltas = {(2,0,1,-2, 0,-1):'MM',
          (0,2,1, 0,-2,-1):'CC',
          (1,1,1,-1,-1,-1):'MC',
          (1,0,1,-1, 0,-1):'M',
          (0,1,1, 0,-1,-1):'C'}

import doctest
class TestCannibals: """
>>> csuccessors([3,3,1,0,0,0])
{(2, 3, 0, 1, 0, 1): 'M->', (3, 1, 0, 0, 2, 1): 'CC->', (3, 2, 0, 0, 1, 1): 'C->', (1, 3, 0, 2, 0, 1): 'MM->', (2, 2, 0
>>> csuccessors([0,0,0,3,3,1])
{(0, 1, 1, 3, 2, 0): '<-C', (0, 2, 1, 3, 1, 0): '<-CC', (1, 0, 1, 2, 3, 0): '<-M', (2, 0, 1, 1, 3, 0): '<-MM', (1, 1, 1
"""
print doctest.testmod() #--> TestResults(failed=0, attempted=2)
print mc_problem()
#--> [(3, 3, 1, 0, 0, 0),
#--> 'CC->',(3, 1, 0, 0, 2, 1),
#--> '<-C', (3, 2, 1, 0, 1, 0),
#--> 'CC->', (3, 0, 0, 0, 3, 1),
#--> '<-C', (3, 1, 1, 0, 2, 0),
#--> 'MM->', (1, 1, 0, 2, 2, 1),
#--> '<-MC', (2, 2, 1, 1, 1, 0),
#--> 'MM->', (0, 2, 0, 3, 1, 1),
#--> '<-C', (0, 3, 1, 3, 0, 0),
#--> 'CC->', (0, 1, 0, 3, 2, 1),
#--> '<-C', (0, 2, 1, 3, 1, 0),
#--> 'CC->', (0, 0, 0, 3, 3, 1)]
```

# 2 Some Other Section

## Finite State Machine (FSM)

A visual representation or a pictorial equivalent to regular expressions. A **non-deterministic FSM** includes epsilon transitions or ambiguity. A **deterministic FSM/lock-step FSM** includes epsilon edges or ambiguity. However, every non-deterministic FSM has a corresponding deterministic FSM that accepts exaclty the same strings. Non-deterministic FSMs are not more powerful, they are just more convenient.

## 2.1 Aspect-oriented Programming

Separate debugging/efficiency statements and the correctness program.

## Server

A server is a machine optimized for sitting in a closet and hosting files.

# 3 Hyper Text Markup Language (HTML)

Invented by Tim Berners-Lee around 1990 and credited with inventint the world-wide web.

Use the tags *strong* and *em* when the contents of your page requires that certain words or phrases be stressed. If you are only highlighting for visual effect use the tags *b* and *i*.

## 3.1 HTTP Request

* Cookie: user_id = 12345; last_seen = Dec 25, 1985

### 3.1.1 GET Request

GET requests are often used for fetching documents and GET parameters are usually used to describe which document we are looking for or what page we are on, etc. They are included in the URL and are ok to cache, should not be used to change the server and are affected by the maximum URL length. GET request: GET /foo HTTP/1.1

### 3.1.2 POST Request

POST parameters are included in the request body, have no max length and are often used for updating data. They are almost never cached.

## 3.2 HTTP Response

A response can be static, which is a pre-written file or dynamic, which is a page made on the fly by programs called web applications.

* Response: HTTP/1.1 200 OK
* Date: Tue Mar 2012 04:33:33 GMT
* Server: Apache /2.2.3 - Similar to User-Agent header on the request. Best to make this up, otherwise you're just giving away free information to a would be hacker that want to know what vulnerability that works against you.
* Content-Type: text/html; charset=utf-8
* Content-Length: 1539
* Set-Cookie: user_id = 12345; Domain = www.example.com; Path = /foo
* Set-Cookie: last_seen = Dec 25, 1985

### Status Codes

200 OK 302 Found - The document is located somewhere else 404 Not Found 500 Server Error - The server broke trying to handle your request

### HTML Header

Valid headers, such as *User-Agent*, *Host*, but really, you can make up all the headers you want.

Use the *<br />* tag instead of *<br>* for an inline line break, but the *p* tag to make a block.

Use the *span* for an inline container, which content can be styled, and *div* for a block container.

If your browser crashes, you should quit using Internet Explorer.

# Cryptography

## Shannon's Keyspace Theorem

## Monoalphabetic Substitution Cipher

Each letter in the alphabet is mapped to a substitution letter.

**CT only attack**. "E" is the most common letter in the English language (appears about 12.7% of the time), which will appear as the most frequent coded letter in the cipher. Next is "T" (9.1%), "A" (8.1%), etc. Next step is to study frequency of pairs of letters: "he", "an", "in", etc.

One way to prove that the cipher is imperfect, is to use *Shannon's Keyspace Theorem*3.2. Assume a 26-letter alphabet,

$$|K| < |M|, 26! < 2^{89}, 26! < 26^{19}. \tag{1}$$

Another way to prove the cipher's imperfection is by showing a ciphertext $c$ that could not decrypt to a message $m$ for any key $k$,

$$c = \mathsf{aa}, m = \mathsf{cs} \tag{2}$$

# Randomness

**Kolmogorov Complexity**. $K(s)$ =length of the shortest possible description of s [Andrey Kolmogorov (1903-1987)]. If there isn't any short program that can describe the sequence, that's an indication that the sequence is random, e.i, $s$ is random if $K(s) = |s| + C$. There's no simplier way to understand the sequence other than to see the whole sequence. However, the Kolmogorov Complexity is **uncomputable**.

**Statistical Tests** - can only show non-randomness. We can always find some non-random sequence that satisfies all of our statistical tests.

**Physically Random Events**

* Quantum Mechanics
* Thermal Noise
* User key presses/mouse movements (?)

**Pseudo-Random Number Generator (PRNG)** takes a small amount of physical randomness and turn them into a long sequence of apparently "random" bits. This can be done by extracting a seed once from a *random pool* and reusing it in every step, encrypting a sequence of values (which can be a counter).

Does this produce a sequence that appears random? No, it repeats values too infrequently, why the key is changed every few million outputs. Another concern is whether the pool of randomness is good enough. On unix machines, this pool is stored in */dev/random* and is collecting events that are believed to be random, like user interactions. A popular PRNG is *Fortuna*.

## 3.3 Secret Sharing

Share a 100-bit long secret among 4 people requires 300 key bits.

$$\begin{aligned} A &: m \oplus k_1 \oplus k_2 \oplus k_3 \\ B &: k_1 \\ C &: k_2 \\ D &: k_3 \end{aligned} \tag{3}$$

# Cipher Block Chaning (CBC) Mode

Assuming $E$ has perfect secrecy (impossible since $|K| \geq |M|$), an attacker can still learn the length of the message and which blocks in $m$ are equal from a captured $c$, where

$$c_i = E_k(m_i) \tag{4}$$

With CBC,

$$\begin{aligned} c_0 &= E_k(m_0 \oplus IV) \\ c_i &= E_k(m_i \oplus c_{i-1}) \end{aligned} \tag{5}$$

where the initial message block is xor:ed with an initialization vector, $IV$, which should not be repeated. The point with the initialization vector is just to hide repetition in the first block. Being lost, the whole message can still be recovered, except for the very first block.

$$\begin{aligned} m_0 &= D_k(c_0) \oplus IV \\ m_{n-i} &= D_k(c_{n-1}) \oplus c_{n-2} \end{aligned} \tag{6}$$

* Requires the encryption function to be invertable
* Does not need the IV to be kept secret, used like another cipher text block. Important is just to not reuse the IV
* Does not provide any protection against tampering
* The final cipher text block depends on all message blocks

## 3.4 Lexical Analysis - Lexing

Break something down into words. A *token* is the smallest unit of lexical analysis output.

| | |
|---|---|
| LANGLE | < |
| LANGLESLASH | </ |
| RANLGE | > |
| EQUAL | = |
| STRING | ”google.com” |
| WORD | Welcome! |

A *Lexical Analyzer* or *lexer* is a collection of token definitions, with the first listed is the winner.

```
import ply.lex as lex

tokens = ('LANGLE', #<
          'LANGLESLASH', #</
          'RANGLE', #>
          'EQUAL', #=
          'STRING', #"hello"
          'WORD') #Welcome!

t_ignore = ' ' #shortcut for whitespace

def t_newline(token):
  r'\n'
  token.lexer.lineno += 1
  pass

def t_error(t):
  print "Lexer: unexpected character " + t.value[0]
  t.lexer.skip(1)

def t_LANGLESLASH(token):
  r'</'
  return token

def t_LANGLE(token):
  r'<'
  return token

def t_RANGLE(token):
  r'>'
  return token
```

```
def t_EQUAL(token):
  r'='
  return token

#def t_NUMBER(token):
#  r'-?\d+(?:\.\d*)?'
#  token.value = float(token.value)
#  return token

def t_STRING(token):
  r'"[^"]*"'
  token.value = token.value[1:-1]
  return token

#def t_WHITESPACE(token):
#  r' '
#  pass

def t_WORD(token):
  r'[^ <>\n]+'
  return token

webpage = """This is
   <b>wepage!
"""
htmllexer = lex.lex()
htmllexer.input(webpage)
while True:
  tok = htmllexer.token()
  if not tok: break
  print tok
```

A state can be either *exclusive* or *inclusive*.

```
states = [('htmlcomment','exclusive')]

def t_htmlcomment(token):
  r'<!--'
  token.lexer.begin('htmlcomment')

def t_htmlcomment_end(token):
  r'-->'
  token.lexer.lineno += token.value.count('\n')
  token.lexer.begin('INITIAL')

def t_htmlcomment_error(token):
  "Gathers up all of the text into one big value so one can count the new lines later."
  token.lexer.skip(1)
```

## 3.5 Syntactical Analysis - Parsing

Breaking down a list of tokens to see if it's valid in the grammar or breaking down a list of words to see if they follow the rules of a language.

A *parsing state* is a rewrite rule from the grammar, augmented with one red dot on the right hand side of the rule.

```
import ply.lex as lex
import ply.yacc as yacc

tokens = (
  'COMMA',
  'IDENTIFIER',
  'LPAREN',
  'NUMBER',
  'RPAREN',
)

def t_NUMBER(t):
  r'-?[0-9]+(\.[0-9]*)?'
  t.value = float(t.value)
  return t

t_COMMA = r','
t_IDENTIFIER = r'[A-Za-z][A-Za-z_]*'
t_LPAREN = r'\('
t_RPAREN = r'\)'

def t_error(t):
  print "JavaScript Lexer: Illegal character " + t.value[0]
  t.lexer.skip(1)

def p_exp_call(p):
    'exp : IDENTIFIER LPAREN optargs RPAREN'
    p[0] = ("call", p[1], p[3])

def p_exp_number(p):
    'exp : NUMBER'
    p[0] = ("number", p[1])

def p_optargs(p):
    'optargs : args'
    p[0] = p[1]

def p_optargs_empty(p):
    'optargs : '
    p[0] = []

def p_args(p):
    'args : exp COMMA args'
    p[0] = [p[1]] + p[3]

def p_args_last(p):
    'args : exp'
    p[0] = [p[1]]

def p_error(p):
    print "Syntax error in input!"

lexer = lex.lex()
parser = yacc.yacc()
print parser.parse("myfun(11,12)",lexer=lexer)
#--> ('call', 'myfun', [('number', 11.0), ('number', 12.0)])
```

## 3.6 JavaScript

*Identifier* are *variable* names or *function* names.

```
<script type="text/javascript">
  function factorial(n){
    if (n==0){
      return 1;
    };
    return n * factorial(n-1);
  }
  document.write(factorial(5));
</script>
```

```
def t_eolcomment(token):
  r'//[^\n]*'
  pass

def t_IDENTIFIER(token):
  r'[a-zA-Z][a-zA-Z_]*'
  return token
```

# Counter (CTR) Mode

The IV is usually devided into a *nonce* and the counter in 64-blocks each (for AES).

$$\begin{aligned} c_i &= E_k(\text{nonce}||i) \oplus m_i \\ m_i &= c_i \oplus E_k(\text{nonce}||i) \end{aligned} \tag{7}$$

* The encryption function does not need to be invertable
* The cipher text is a little longer than the message
* If you encrypt the samme message twice, you will get different ciphertexts
* Parallelizable (unlike CBC). The encryption function does not depend on the message and can be computed in advance. if you have 3 AES engines encryption will work 3 times as fast
* Does not need padding
* In every single aspect CTR Mode dominates CBC and is the recommended mode to be used today

# Cipher Feedback (CFB) Mode

Uses an additional parameter $s < n$, which is the size of the message block that is less than the normal block size of the cipher.

$$\begin{aligned} x_0 &= IV \\ x_i &= x_{i-1}[s:]||c_{i-1} \\ c_i &= E_k(x_i)[:s] \oplus m_i \end{aligned} \tag{8}$$

Decryption

$$\begin{aligned} m_i &= c_i \oplus E_k(x_i)[:s] \\ x_i &= x_{i-1}[s:]||c_i \\ x_0 &= IV \end{aligned} \tag{9}$$

* Does not require the encryption function to be invertable
* Does not need the IV to be kept secret, used like another cipher text block. Important is just to not reuse the IV.
* Can use small message blocks, by only encrypt the message in chunks of size s. Turns the block cipher into a stream cipher
* Does not provide any protection against tampering
* The final cipher text block depends on all message blocks

**Outline** The remainder of this article is organized as follows. Section **??** gives account of previous work. Our new and exciting results are described in Section **??**. Finally, Section **??** gives the conclusions.

# 4 Google App Engine

## 4.1 Handlers

```
class TestHandler(webapp2.RequestHandler):
  def get(self):
    q = self.request.get("q") #get parameter q
    self.response.out.write(q)

app = webapp2.WSGIApplication([('/',MainPage),('/testform',TestHandler)],debug=True)
```

## 4.2 Forms

```
form="""
  <form method="post">
  <label>Free Field <input name="q" value="%(q)s"></label>
  <div style="color: red">%(error)s</div>
  </form>
"""
def validation_function(q):
  reutrn ...

class MainPage(webapp2.RequestHandler):
  def write_form(self,error="",q=""):
    self.response.out.write(form % {'error':error,'q':q})

  def get(self):
    self.write_form()

  def post(self):
    user_q = self.request.get('q')
    q      = validation_function(user_q)

    if not q:
      self.write_form("Invalid form.",user_q)
    else:
      self.response.out.write("Thanks! That's totally valid!")
```

User input need to be esacped.

```
import cgi
print cgi.escape('"<b&ld>"',quote=True)
#--> &quot;&lt;b&amp;ld&gt;&quot;
```

### 4.2.1 Redirection

With redirection one can reload the page without having resubmitting a form. It's also good practice to have distinct pages for forms and successes.

## 4.3 Templates

Install the *jinja2* library
sudo easy_install jinja2
and modify your *app.yaml* file

```
application: <username>
version: 1
runtime: python27
api_version: 1
threadsafe: true

libraries:
- name: jinja2
  version: latest

handlers:
- url: /.*
  script: asciichan.app
```

Make a simple html file in your *templates* directory

```
<!DOCTYPE html>
<html>
  <head>
    <title>/ascii/</title>
  </head>

  <body>
    <h1>/ascii/</h1>
  </body>
</html>
```

and write your main file

```
import os
import webapp2
import jinja2

template_dir = os.path.join(os.path.dirname(__file__), 'templates')
jinja_env = jinja2.Environment(loader = jinja2.FileSystemLoader(template_dir), autoescape = True)

class Handler(webapp2.RequestHandler):
  def write(self,*a,**kw):
    self.response.out.write(*a,**kw)

  def render_str(self,template,**params):
    t = jinja_env.get_template(template)
    return t.render(params)

  def render(self,template,**kw):
```

```
    self.write(self.render_str(template,**kw))

class MainPage(Handler):
  def get(self):
      self.render('front.html')

app = webapp2.WSGIApplication([('/',MainPage)],debug=True)
```

## 4.4 CSS

```
application: <username>
version: 1
runtime: python27
api_version: 1
threadsafe: true

handlers:
- url: /static/
  static_dir: static

- url: /.*
  script: asciichan.app
```

## 4.5 Google App Engine Datastore

**Entity.** Tables, where the columns aren't fixed, all have id fields and have a notion of parents/ancestors which is a relation to other entities.

### 4.5.1 GQL

A simplified version of SQL, where all queries begins with *select **, there are no joins and all queries must be indexed.

```
posts = db.GqlQuery("select * from Post order by created desc")
posts = Post.all().order('-created') #same
```

### 4.5.2 Types

* Integer
* Float
* String - < 500 chars, can be indexed and sorted
* Text - > 500 chars, cannot be indexed or sorted
* Date
* Time
* DateTime
* Email
* Link
* PostalAddress

```
from google.appengine.ext import db
class Post(db.Model):
  subject = db.StringProperty(required       = True)
  content = db.TextProperty(required         = True)
  created = db.DateTimeProperty(auto_now_add = True)
  updated = db.DateTimeProperty(auto_now     = True)
```

## 4.6 Cookies

A small piece of (temporary) data, clientside enforced stored in the browser for a website. Generally one can store about 20 cookies per website, which is a browser limitation. The length of the cookie is limited to around 4 kb. It also has to be a direct match or a subset to a particular domain.

Good uses of cookies

* Storing login information
* Storing small amounts of data to avoid hitting a database
* tracking a user for ads
* storing user preferences info - NO, want data to survice

One can change a cookie within the console in one's browser's development tools

document.cookie # "visits=6"

document.cookie="visits=10000"

Secure the cookie with a HMAC

```
import os
import webapp2
import jinja2

from google.appengine.ext import db

template_dir = os.path.join(os.path.dirname(__file__), 'templates')
jinja_env = jinja2.Environment(loader = jinja2.FileSystemLoader(template_dir), autoescape = True)

#-----------------------------------------
#import hashlib
import hmac
SECRET = 'secret'

def check_secure_val(h):
  val = h.split('|')[0]
  if h == make_secure_val(val): return val

def hash_str(s):
  #return hashlib.md5(s).hexdigest()
  return hmac.new(SECRET,s).hexdigest()

def make_secure_val(s):
  return "%s|%s" % (s,hash_str(s))
#-----------------------------------------

class Handler(webapp2.RequestHandler):
  def write(self,*a,**kw):
    self.response.out.write(*a,**kw)

  def render_str(self,template,**params):
    t = jinja_env.get_template(template)
    return t.render(params)

  def render(self,template,**kw):
    self.write(self.render_str(template,**kw))

class MainPage(Handler):
  def get(self):
    self.response.headers['Content-Type'] = 'text/plain'
    visits = 0
    visit_cookie_str = self.request.cookies.get('visits')
```

```
    if visit_cookie_str:
      cookie_val = check_secure_val(visit_cookie_str)
      if cookie_val:
        visits = int(cookie_val)
    visits += 1
    new_cookie_val = make_secure_val(str(visits))
    self.response.headers.add_header('Set-Cookie','visits=%s' % new_cookie_val)
    self.write('You\'ve been here %s times' % visits)


app = webapp2.WSGIApplication([('/',MainPage)],debug=True)
```

document.cookie # "visits=6—295c82aceeb5f3715e6e6e3304199e1ae0"

# 5 Qotes

Ascii art. It's a fantastic way for people to waste their time in front of their keyboards. - Hoffman

If your data has structure, use Oracle. If your data has no structure, use Hadoop. If your data has no value, use MongoDB.

Time flies like an arrow, fruit flies like a banana.

# Index