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

April 27, 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

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'
```

1.5 itertools

1.5.1 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
```

1.6 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]
```

1.7 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
```

1.8 The Law of Diminishing Returns

1.9 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
```

1.10 Substring

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

1.11 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
```

1.12 Translation Table

```
import string

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

1.13 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
```

1.14 Regular Expressions

The module to import in python is called re. A regular expression is written

1.14.1 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.

1.14.2 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>
```

\b	word boundary
*	Kleene Operator
+	Kleene Operator

1.14.3 split

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

1.15 sub

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

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

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

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

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 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).

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

* 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.

Index