Overview

What is COS 126? Broad, but technical, intro to CS.

Goals.
- Demystify computer systems.
- Empower you to exploit available technology.
- Build awareness of substantial intellectual underpinnings.

Topics.
- Programming in Java.
- Machine architecture.
- Theory of computation.
- Applications to science, engineering, and commercial computing.

“Computers are incredibly fast, accurate, and stupid; Humans are incredibly slow, inaccurate, and brilliant; together they are powerful beyond imagination.”
The Basics

Lectures. [Kevin Wayne]
- Tuesdays and Thursdays, Frist 302.
- Same lecture at 10am and 11am.

Precepts. [Donna Gabai (lead) · Aditya Bhaskara · Will Clarkson · Rob Dockins · Michael Golightly · Thomas Mason · Chris Park · JP Singh]
- Tue+Thu or Wed+Fri.
- Tips on assignments, worked examples, clarify lecture material.

Friend 016/017 lab. [Undergrad lab assistants]
- Weekdays 7-11pm, some weekend hours.
- Full schedule on Web.

For full details: See www.princeton.edu/~cos126
Grades

Course grades. No preset curve or quota.

9 programming assignments. 40%.
2 exams. 50%.
Final programming project. 10%.
Extra credit and staff discretion. Adjust borderline cases.

participation helps, frequent absences hurts
Course Materials

Course website. [www.princeton.edu/~cos126]
- Submit assignments, check grades.
- Programming assignments.
- Lecture slides.

print before lecture; annotate during lecture

skim before lecture; read thoroughly afterwards

Required readings. Sedgewick and Wayne. *Intro to Programming in Java: An Interdisciplinary Approach.* [Labyrinth Books]

Recommended readings. Harel. *What computers can't do.* [Labyrinth]
Programming Assignments

Desiderata.
- Address an important scientific or commercial problem.
- Illustrate the importance of a fundamental CS concept.

Examples.
- N-body simulation.
- Pluck a guitar string.
- DNA sequence alignment.
- Estimate Avogadro's number.

Due. Mondays 11pm via Web submission.

Computing equipment.
- Your laptop. [OS X, Windows, Linux, iPhone, ... ]
- OIT desktop. [Friend 016 and 017 labs]
What's Ahead?

Lecture 2. Intro to Java.

Precept 1. Meets today/tomorrow.
Precept 2. Meets Thu/Fri.

Not registered? Go to any precept now; officially register ASAP.
Change precepts? Use SCORE.

Assignment 0. Due Monday, 11pm.
  - Read Sections 1.1 and 1.2 in textbook.
  - Install Java programming environment + a few exercises.
  - Lots of help available, don't be bashful.

END OF ADMINISTRATIVE STUFF
0. Prologue: A Simple Machine
Secure Chat

Alice wants to send a secret message to Bob?

- Can you read the secret message gX76W3v7K?
- But Bob can. How?
Encryption Machine

**Goal.** Design a machine to encrypt and decrypt data.

Encipher:

```
SENDMONEY
```

decipher:

```
GX76W3v7K
```

**Enigma encryption machine.**

- "Unbreakable" German code during WWII.
- Broken by Turing bombe.
- One of first uses of computers.
- Helped win Battle of Atlantic by locating U-boats.
A Digital World

Data is a sequence of bits.
- Text.
- Documents, pictures, sounds, movies, …
- Programs, executables.

File formats.  txt, pdf, doc, ppt, jpeg, mp3, divx, java, exe, …
A Digital World

Data is a sequence of bits.
- Text.
- Documents, pictures, sounds, movies, ...
- Programs, executables.

Base64 encoding. Use 6 bits to represent each alphanumerical symbol.

<table>
<thead>
<tr>
<th>Binary</th>
<th>Char</th>
<th>Binary</th>
<th>Char</th>
<th>Binary</th>
<th>Char</th>
<th>Binary</th>
<th>Char</th>
<th>Binary</th>
<th>Char</th>
<th>Binary</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000</td>
<td>A</td>
<td>001011</td>
<td>L</td>
<td>010110</td>
<td>W</td>
<td>100001</td>
<td>h</td>
<td>101100</td>
<td>s</td>
<td>110111</td>
<td>3</td>
</tr>
<tr>
<td>000001</td>
<td>B</td>
<td>001100</td>
<td>M</td>
<td>010111</td>
<td>X</td>
<td>100010</td>
<td>i</td>
<td>101101</td>
<td>t</td>
<td>111000</td>
<td>4</td>
</tr>
<tr>
<td>000010</td>
<td>C</td>
<td>001101</td>
<td>N</td>
<td>011000</td>
<td>Y</td>
<td>100111</td>
<td>j</td>
<td>101110</td>
<td>u</td>
<td>111001</td>
<td>5</td>
</tr>
<tr>
<td>000011</td>
<td>D</td>
<td>001110</td>
<td>O</td>
<td>011001</td>
<td>Z</td>
<td>100100</td>
<td>k</td>
<td>101111</td>
<td>v</td>
<td>111010</td>
<td>6</td>
</tr>
<tr>
<td>000100</td>
<td>E</td>
<td>001111</td>
<td>P</td>
<td>011010</td>
<td>a</td>
<td>101010</td>
<td>l</td>
<td>110000</td>
<td>w</td>
<td>111011</td>
<td>7</td>
</tr>
<tr>
<td>000101</td>
<td>F</td>
<td>010000</td>
<td>Q</td>
<td>011011</td>
<td>b</td>
<td>101110</td>
<td>m</td>
<td>110001</td>
<td>x</td>
<td>111100</td>
<td>8</td>
</tr>
<tr>
<td>000110</td>
<td>G</td>
<td>010001</td>
<td>R</td>
<td>011100</td>
<td>c</td>
<td>101111</td>
<td>n</td>
<td>110010</td>
<td>y</td>
<td>111101</td>
<td>9</td>
</tr>
<tr>
<td>000111</td>
<td>H</td>
<td>010010</td>
<td>S</td>
<td>011101</td>
<td>d</td>
<td>101000</td>
<td>o</td>
<td>110011</td>
<td>z</td>
<td>111110</td>
<td>+</td>
</tr>
<tr>
<td>001000</td>
<td>I</td>
<td>010011</td>
<td>T</td>
<td>011110</td>
<td>e</td>
<td>101001</td>
<td>p</td>
<td>110100</td>
<td>0</td>
<td>111111</td>
<td>/</td>
</tr>
<tr>
<td>001001</td>
<td>J</td>
<td>010100</td>
<td>U</td>
<td>011111</td>
<td>f</td>
<td>101010</td>
<td>q</td>
<td>110101</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>001010</td>
<td>K</td>
<td>010101</td>
<td>V</td>
<td>100000</td>
<td>g</td>
<td>101011</td>
<td>r</td>
<td>110110</td>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
One-Time Pad Encryption

**Encryption.**

- Convert text message to N bits.

0 or 1

---

**Base64 Encoding**

<table>
<thead>
<tr>
<th>char</th>
<th>dec</th>
<th>binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>000000</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>000001</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Y</td>
<td>24</td>
<td>011000</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

---

message

010010 000100 001101 000011 001100 001110 001101 000100 011000

base64
One-Time Pad Encryption

**Encryption.**
- Convert text message to N bits.
- Generate N random bits (one-time pad).

<table>
<thead>
<tr>
<th>S</th>
<th>E</th>
<th>N</th>
<th>D</th>
<th>M</th>
<th>O</th>
<th>N</th>
<th>E</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>010010</td>
<td>000100</td>
<td>001101</td>
<td>000011</td>
<td>001100</td>
<td>001110</td>
<td>001101</td>
<td>000100</td>
<td>011000</td>
</tr>
<tr>
<td>110010</td>
<td>010011</td>
<td>110110</td>
<td>111001</td>
<td>011010</td>
<td>111001</td>
<td>100010</td>
<td>111111</td>
<td>010010</td>
</tr>
</tbody>
</table>

message  
base64  
random bits
One-Time Pad Encryption

Encryption.
- Convert text message to N bits.
- Generate N random bits (one-time pad).
- Take bitwise XOR of two bitstrings.

![XOR Truth Table](image)

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>x ^ y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

sum corresponding pair of bits: 1 if sum is odd, 0 if even

<table>
<thead>
<tr>
<th>S</th>
<th>E</th>
<th>N</th>
<th>D</th>
<th>M</th>
<th>O</th>
<th>N</th>
<th>E</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>010010</td>
<td>000100</td>
<td>001101</td>
<td>000011</td>
<td>001100</td>
<td>001110</td>
<td>001101</td>
<td>000100</td>
<td>011000</td>
</tr>
<tr>
<td>110010</td>
<td>010011</td>
<td>110110</td>
<td>111001</td>
<td>011010</td>
<td>111001</td>
<td>100010</td>
<td>111111</td>
<td>010010</td>
</tr>
<tr>
<td>100000</td>
<td>010111</td>
<td>111011</td>
<td>111010</td>
<td>010110</td>
<td>110111</td>
<td>101111</td>
<td>111011</td>
<td>001010</td>
</tr>
</tbody>
</table>

0 ^ 1 = 1
One-Time Pad Encryption

Encryption.
- Convert text message to N bits.
- Generate N random bits (one-time pad).
- Take bitwise XOR of two bitstrings.
- Convert binary back into text.

Base64 Encoding

<table>
<thead>
<tr>
<th>char</th>
<th>dec</th>
<th>binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>000000</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>000001</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>X</td>
<td>23</td>
<td>010111</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

message

base64

random bits

XOR

encrypted
Secure Chat

Alice wants to send a secret message to Bob?

- Can you read the secret message $gX76W3v7K$?
- But Bob can. How?
One-Time Pad Decryption

Decryption.
  - Convert encrypted message to binary.
One-Time Pad Decryption

Decryption.

- Convert encrypted message to binary.

```
<table>
<thead>
<tr>
<th>g</th>
<th>X</th>
<th>7</th>
<th>6</th>
<th>W</th>
<th>3</th>
<th>v</th>
<th>7</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>100000</td>
<td>010111</td>
<td>111011</td>
<td>111010</td>
<td>010110</td>
<td>110111</td>
<td>101111</td>
<td>111011</td>
<td>001010</td>
</tr>
</tbody>
</table>
```

Base64 Encoding

<table>
<thead>
<tr>
<th>char</th>
<th>dec</th>
<th>binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>000000</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>000001</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>X</td>
<td>23</td>
<td>010111</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
One-Time Pad Decryption

Decryption.
- Convert encrypted message to binary.
- Use same N random bits (one-time pad).

<table>
<thead>
<tr>
<th>g</th>
<th>X</th>
<th>7</th>
<th>6</th>
<th>W</th>
<th>3</th>
<th>v</th>
<th>7</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>100000</td>
<td>010111</td>
<td>111011</td>
<td>111010</td>
<td>010110</td>
<td>110111</td>
<td>101111</td>
<td>111011</td>
<td>001010</td>
</tr>
<tr>
<td>110010</td>
<td>010011</td>
<td>110110</td>
<td>111001</td>
<td>011010</td>
<td>111001</td>
<td>100010</td>
<td>111111</td>
<td>010010</td>
</tr>
</tbody>
</table>

encrypted base64
random bits
One-Time Pad Decryption

Decryption.
- Convert encrypted message to binary.
- Use same N random bits (one-time pad).
- Take bitwise XOR of two bitstrings.

XOR Truth Table

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>x ^ y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>g</th>
<th>X</th>
<th>7</th>
<th>6</th>
<th>W</th>
<th>3</th>
<th>v</th>
<th>7</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>100000</td>
<td>010111</td>
<td>111011</td>
<td>111010</td>
<td>010110</td>
<td>110111</td>
<td>101111</td>
<td>111011</td>
<td>001010</td>
</tr>
<tr>
<td>110010</td>
<td>010011</td>
<td>110110</td>
<td>110001</td>
<td>011010</td>
<td>111001</td>
<td>100010</td>
<td>111111</td>
<td>010010</td>
</tr>
<tr>
<td>010010</td>
<td>000100</td>
<td>001101</td>
<td>000011</td>
<td>001100</td>
<td>001110</td>
<td>001101</td>
<td>000100</td>
<td>011000</td>
</tr>
</tbody>
</table>

encrypted

base64

random bits

XOR

1 ^ 1 = 0
One-Time Pad Decryption

**Decryption.**

- Convert encrypted message to binary.
- Use same N random bits (one-time pad).
- Take bitwise XOR of two bitstrings.
- Convert back into text.

### Base64 Encoding

<table>
<thead>
<tr>
<th>char</th>
<th>dec</th>
<th>binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>000000</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>000001</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Y</td>
<td>24</td>
<td>011000</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

```
encrypted
\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|
base64
\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|
random bits
\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|
XOR
\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|
message
\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|\|
```
Why Does It Work?

**Crucial property.** Decrypted message = original message.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>original message bit</td>
</tr>
<tr>
<td>b</td>
<td>one-time pad bit</td>
</tr>
<tr>
<td>^</td>
<td>XOR operator</td>
</tr>
<tr>
<td>a ^ b</td>
<td>encrypted message bit</td>
</tr>
<tr>
<td>(a ^ b) ^ b</td>
<td>decrypted message bit</td>
</tr>
</tbody>
</table>

Why is crucial property true?

- Use properties of XOR.
- \((a ^ b) ^ b = a ^ (b ^ b) = a ^ 0 = a\)

**XOR Truth Table**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y</td>
<td>x ^ y</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>-------</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
One-Time Pad Decryption (with the wrong pad)

Decryption.
- Convert encrypted message to binary.
- Use wrong N bits (bogus one-time pad).
- Take bitwise XOR of two bitstrings.
- Convert back into text: *Oops.*

<table>
<thead>
<tr>
<th>g</th>
<th>X</th>
<th>7</th>
<th>6</th>
<th>W</th>
<th>3</th>
<th>v</th>
<th>7</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>100000</td>
<td>010111</td>
<td>111011</td>
<td>110100</td>
<td>010110</td>
<td>110111</td>
<td>101111</td>
<td>111011</td>
<td>001010</td>
</tr>
<tr>
<td>101000</td>
<td>011100</td>
<td>110101</td>
<td>101111</td>
<td>010010</td>
<td>111001</td>
<td>100101</td>
<td>101010</td>
<td>001010</td>
</tr>
<tr>
<td>001000</td>
<td>001011</td>
<td>001110</td>
<td>010101</td>
<td>000100</td>
<td>001110</td>
<td>001010</td>
<td>010001</td>
<td>000000</td>
</tr>
<tr>
<td>E</td>
<td>O</td>
<td>V</td>
<td>E</td>
<td>O</td>
<td>K</td>
<td>R</td>
<td>A</td>
<td></td>
</tr>
</tbody>
</table>

encrypted

base64

wrong bits

XOR

wrong message
Goods and Bads of One-Time Pads

**Good.**
- Very simple encryption/decryption processes.
- Provably unbreakable if pad is truly random. [Shannon, 1940s]

  eavesdropper Eve sees only random bits

**Bad.**
- Easily breakable if pad is re-used.
- Pad must be as long as the message.
- Truly random bits are very hard to come by.
- **Pad must be distributed securely.**

  impractical for Web commerce
Random Numbers

“Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin.”

Jon von Neumann (left), ENIAC (right)
Pseudo-Random Bit Generator

Practical middle-ground.
- Let’s make a pseudo-random bit generator gadget.
- Alice and Bob each get identical small gadgets.

instead of identical large one-time pads

How to make small gadget that produces "random" numbers.
- Linear feedback shift register.
- Linear congruential generator.
- Blum-Blum-Shub generator.
- ...
Shift register terminology.

- **Bit**: 0 or 1.
- **Cell**: storage element that holds one bit.
- **Register**: sequence of cells.
- **Seed**: initial sequence of bits.
- **Shift register**: when clock ticks, bits propagate one position to left.
A linear feedback shift register is a 11-bit register. The new output bit is calculated as the XOR of bits 8 and 10. The output bit is the same as the least significant bit (bit 0).
Random Numbers

Q. Are these 2000 numbers random? If not, what is the pattern?

A. No. This is output of an 11 bit LFSR!
LFSR Challenge 1

**Goal.** Decrypt/encrypt 1,000 characters. Can we use an 11-bit LFSR?

A. Yes, no problem.

B. No, the bits it produces are not truly random.
   - True, but that's beside the point.

C. No, need a longer LFSR.
   - Only $2^{11}$ bit patterns for register.
   - "Random" bits cycle after $2^{11} - 1 = 2047$ steps.

**Lesson.** LFSR are scalable: 20 cells for 1 million bits; 30 cells for 1 billion. (but need theory of finite groups to know where to put taps)
LFSR Challenge 2

Goal. Decrypt/encrypt 1 gigabyte movie. How big an LFSR?

A. 30 bits should be enough.
   - Probably not; Eve can try all $2^{30}$ possibilities and see which one results in a movie.

B. 100 bits is safe.
   - Maybe; Eve would need $10^{12}$ centuries to try all $2^{100}$ possibilities.

C. 1000 bits makes it sufficiently secure.
   - Experts have cracked LFSR.
   - More complicated machines needed.
Other LFSR Applications

What else can we do with a LFSR?

- DVD encryption with CSS.
- DVD decryption with DeCSS!
- Subroutine in military cryptosystems.

```c
/* efdtt.c     Author: Charles M. Hannum <root@ihack.net> */
/* Usage is: cat title-key scrambled.vob | efdtt >clear.vob */
#define m(i)(x[i]^s[i+84])<<

unsigned char x[5] ,y,s[2048];main(n){for( read(0,x,5 );read(0,s ,n=2048 ) ; write(1 ,s,n) )if(s[y=s[13]%8+20]/16%4==1){int i=m(1)17 ^256 +m(0) 8,k =m(2)
0,j= m(4) 17^ m(3) 9^k* 2^-k%8
^8,a =0,c =26;for (s[y] -=16;
--c;j *2)a= a*2^i& 1,i=i /2^j&1
<<24;for(j= 127; ++j<n;c=c>
y)
c
+=y=i^i/8^i>>4^i>>12,
i=i>>8^y<<17,a^=a>>14,y=a^a*8^a<<6,a=a
>>8^y<<9,k=s[j],k ="7Wo~'G_\216"[k &7]+2^"cr3sfw6v;*k+>/n."[k>>4]*2^k*257/8,s[j]=k^(k&k*2&34)*6^c+-y
;}}
```

http://www.cs.cmu.edu/~dst/DeCSS/Gallery
LFSR and "General Purpose Computer"

Important properties.
- Built from simple components.
- Scales to handle huge problems.
- Requires a deep understanding to use effectively.

<table>
<thead>
<tr>
<th>Basic Component</th>
<th>LFSR</th>
<th>Computer</th>
</tr>
</thead>
<tbody>
<tr>
<td>control</td>
<td>start, stop, load</td>
<td>same</td>
</tr>
<tr>
<td>clock</td>
<td>regular pulse</td>
<td>2.8 GHz pulse</td>
</tr>
<tr>
<td>memory</td>
<td>11 bits</td>
<td>1 GB</td>
</tr>
<tr>
<td>input</td>
<td>seed</td>
<td>sequence of bits</td>
</tr>
<tr>
<td>computation</td>
<td>shift, XOR</td>
<td>logic, arithmetic, ...</td>
</tr>
<tr>
<td>output</td>
<td>pseudo-random bits</td>
<td>Sequence of bits</td>
</tr>
</tbody>
</table>

Critical difference. General purpose machine can be programmed to simulate ANY abstract machine.
A Profound Idea

Programming. Can write a Java program to simulate the operations of any abstract machine.

- Basis for theoretical understanding of computation. [stay tuned]
- Basis for bootstrapping real machines into existence. [stay tuned]

Stay tuned. See Assignment 5.

```java
public class LFSR {
    private int fill[];
    private int tap;
    private int N;

    public LFSR(String fill, int tap) { ... }

    public int step() { ... }

    public static void main(String[] args) {
        LFSR lfsr = new LFSR("01101000010", 8);
        for (int i = 0; i < 2000; i++)
            StdOut.println(lfsr.step());
    }
}
```

% java LFSR
11001001001111011011100101101
011100110001011111010010001
00110100101111001100100111...

A Profound Question

Q. What is a random number?

LFSR does not produce random numbers.
- It is a very simple deterministic machine.
- But it is hard to distinguish the bits it produces from random ones.

Q. Are random processes found in nature?
- Motion of cosmic rays or subatomic particles?
- Mutations in DNA?

Q. Is the natural world a (not-so-simple) deterministic machine?

“God does not play dice.” — Albert Einstein