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.

[$SEND\ MONEY$]

encrypt

g $X\ 7\ 6\ W\ 3\ v\ 7\ K$

decrypt

[$SEND\ MONEY$]
**Encryption Machine**

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

```
SENDMONEY
```

encrypt

```
SENDMONEY
```

decrypt

**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. [bit = 0 or 1]
- Text.
- Programs, executables.
- Documents, pictures, sounds, movies, ...

**One-Time Pad Encryption**

Encryption.
- Convert text message to N bits. [0 or 1]

Ex. Base64 encoding of text.
- Simple method for representing A-Z, a-z, 0-9, +, /
- 6 bits to represent each symbol (64 symbols)

```
000000 A 001000 Z 010000 Q 011000 Y 100000 g 101000 o 110000 w 111000 4
000001 B 001001 J 010001 K 011001 r 100001 h 101001 p 110001 x 111001 5
000010 C 001010 R 010010 S 011010 v 100010 f 101010 t 110010 l 111010 6
000011 D 001011 T 010011 u 011011 D 100011 j 101011 f 110011 z 111011 7
000100 E 001100 M 010100 U 011100 c 100100 k 101100 s 110100 0 111100 8
000101 F 001101 N 010101 W 011101 e 100101 l 101101 t 110101 1 111101 9
000110 G 001110 O 010110 N 011110 g 100110 m 101110 u 110110 2 111110 4
000111 H 001111 P 010111 X 011111 f 100111 n 101111 v 110111 3 111111 7
```

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>C</td>
<td>2</td>
<td>000002</td>
</tr>
<tr>
<td>D</td>
<td>3</td>
<td>000003</td>
</tr>
<tr>
<td>E</td>
<td>4</td>
<td>000004</td>
</tr>
<tr>
<td>F</td>
<td>5</td>
<td>000005</td>
</tr>
<tr>
<td>G</td>
<td>6</td>
<td>000006</td>
</tr>
</tbody>
</table>

```
010010 000100 001101 010011 011000 001110 000110 000100 011000
```

message

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>011011</td>
<td>111011</td>
<td>100010</td>
<td>111111</td>
<td>010010</td>
</tr>
</tbody>
</table>

message
base64
random bits

<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>011011</td>
<td>111011</td>
<td>100010</td>
<td>111111</td>
<td>010010</td>
</tr>
</tbody>
</table>

message
base64
one-time pad
XOR

100000 010011 111011 111010 011010 110111 101111 111011 001010

Secure Chat

Alice wants to send a secret message to Bob
• Can you read the secret message gX76W3v7K?
• But Bob can. How?

Bob thing
One-Time Pad Decryption

Decryption.
• Convert encrypted message to binary.
• Use same N random bits (one-time pad).
  - Key point: Bob and Alice agreed on the one-time pad beforehand

<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>g</td>
<td>X</td>
<td>7</td>
<td>6</td>
<td>W</td>
<td>3</td>
<td>v</td>
<td>7</td>
<td>K</td>
</tr>
</tbody>
</table>

encrypted

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>W</td>
<td>22</td>
<td>010110</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>encrypted</th>
</tr>
</thead>
<tbody>
<tr>
<td>100000 010111 111011 111010 010110 110111 101111 111011 001010</td>
</tr>
</tbody>
</table>

encrypted

base64

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>

XOR

g | X | 7 | 6 | W | 3 | v | 7 | K |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>g</td>
<td>X</td>
<td>7</td>
<td>6</td>
<td>W</td>
<td>3</td>
<td>v</td>
<td>7</td>
<td>K</td>
</tr>
</tbody>
</table>

encrypted

random bits

<table>
<thead>
<tr>
<th>encrypted</th>
</tr>
</thead>
<tbody>
<tr>
<td>010010 001001 000011 011000 001010 001100 001010 001000 011000</td>
</tr>
</tbody>
</table>

encrypted

one-time pad

base64

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.

<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>M</td>
<td>12</td>
<td>001100</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Base64 Encoding**

- **encrypted**
  - g
  - X
  - 7
  - 6
  - W
  - 3
  - v
  - 7
  - K

- **base64**
  - 100000
  - 010111
  - 111011
  - 111010
  - 010110
  - 110111
  - 111011
  - 001010

- **one-time pad**
  - 110010
  - 010011
  - 111010
  - 111011
  - 010110
  - 110111
  - 011010

- **XOR**
  - 010010
  - 000100
  - 001101
  - 001100
  - 001101
  - 001110
  - 001101
  - 001100

- **message**
  - S
  - E
  - N
  - D
  - M
  - O
  - N
  - E
  - Y

---

### 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>

**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>

**Why is crucial property true?**
- Use properties of XOR.
  - \((a ^ b) ^ b = a ^ (b ^ b) = a ^ 0 = a\)

**magic?**

---

**One-Time Pad Decryption (with the wrong pad)**

**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>111011</td>
<td>001010</td>
<td></td>
</tr>
</tbody>
</table>

---

**One-Time Pad Decryption (with the wrong pad)**

**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>111011</td>
<td>001010</td>
<td>base64</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>g</td>
<td>X</td>
<td>7</td>
<td>6</td>
<td>w</td>
<td>3</td>
<td>v</td>
<td>7</td>
<td>k</td>
</tr>
<tr>
<td>10000 010111 111011 111010 010110 110111 101111 111011 001010</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10100 011100 110101 101111 010010 110101 101010 001010</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>base64</th>
<th>encrypted</th>
<th>wrong bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000 010111 111011 111010 010110 110111 111011 001010</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10100 011100 110101 101111 010010 110101 101010 001010</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Key point:** Without the pad, Eve cannot understand the message.

But Eve has a computer. Why not try all possible pads?

**One problem:** it might take a long time [stay tuned].

**Worse problem:** she would see all possible messages!
- 54 bits
- $2^{54}$ possible messages, all different.
- $2^{54}$ possible encoded messages, all different.
- No way for Eve to distinguish real message from any other message.

One-time pad is “provably secure”.

**Eve’s Problem (one-time pads)**

Key point: Without the pad, Eve cannot understand the message.

But Eve has a computer. Why not try all possible pads?

One problem: it might take a long time [stay tuned].

Worse problem: she would see all possible messages!
- 54 bits
- $2^{54}$ possible messages, all different.
- $2^{54}$ possible encoded messages, all different.
- No way for Eve to distinguish real message from any other message.

One-time pad is “provably secure”.

<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>g</td>
<td>X</td>
<td>7</td>
<td>6</td>
<td>w</td>
<td>3</td>
<td>v</td>
<td>7</td>
<td>k</td>
</tr>
<tr>
<td>10000 010111 111011 111010 010110 110111 101111 111011 001010</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10100 011100 110101 101111 010010 110101 101010 001010</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>base64</th>
<th>encrypted</th>
<th>wrong bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000 010111 111011 111010 010110 110111 111011 001010</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10100 011100 110101 101111 010010 110101 101010 001010</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>base64</th>
<th>XOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>001000 001011 001110 010101 001110 001010 001110 001010</td>
<td></td>
</tr>
<tr>
<td>001000 001011 001110 010101 001110 001010 001110 001010</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>base64</th>
<th>wrong message</th>
</tr>
</thead>
<tbody>
<tr>
<td>ILOVEOKRA</td>
<td></td>
</tr>
<tr>
<td>ILOVEOKRA</td>
<td></td>
</tr>
<tr>
<td>ILOVEOKRA</td>
<td></td>
</tr>
<tr>
<td>ILOVEOKRA</td>
<td></td>
</tr>
<tr>
<td>ILOVEOKRA</td>
<td></td>
</tr>
<tr>
<td>ILOVEOKRA</td>
<td></td>
</tr>
<tr>
<td>ILOVEOKRA</td>
<td></td>
</tr>
<tr>
<td>ILOVEOKRA</td>
<td></td>
</tr>
</tbody>
</table>
Goods and Bads of One-Time Pads

Good.
• Easily computed by hand.
• Very simple encryption/decryption processes.
• Provably unbreakable if bits are truly random. [Shannon, 1940s]

Bad.
• (After a short break . . .)

COS 126 Overview

What is COS 126? Broad, but technical, introduction to computer science.

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.” — Albert Einstein

The Basics

Lectures. [D. Clark]

Precepts. [Gabai · Pritchard · Beringer · Chaney · S. Clark · Funkhouser · Israel · Kang · Lee · C. Liu · S. Liu · Nadimpalli · Pai · Przytycki · Sepin · Song]
• Tips on assignments; worked examples.
• Questions on lecture material.
• Informal and interactive.

Friend 016/017 lab. [Undergrad assistants]
• Help with systems/debugging, not with course material.
• Full schedule on Web (usually Sun—Fri evenings, Sat. afternoons)
• Starts this week!

Website knows all: www.princeton.edu/~cos126

Grades

Course grades. No preset curve or quota.

9 programming assignments. 40%.
2 written exams (in lecture, 3/12 and 5/7). 35%.
2 programming exams (evenings, 3/14, 5/6). 15%.
Final programming project (due Dean’s date - 1). 10%.
Extra credit / staff discretion. Adjust borderline cases.

Participation helps, frequent absence hurts.
Course Materials

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

Course text.
Sedgewick and Wayne.
Intro to Programming in Java: An Interdisciplinary Approach.

Recommended reading (lectures 19-20).
Harel.
Computers Ltd.: What computers really can’t do.

Programming Assignments

Desiderata.
- Address an important scientific or commercial problem.
- Illustrate the importance of a fundamental CS concept.
- You solve problem from scratch!

Due. Mondays 9 PM via Web submission.

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

Advice.
- Start early: plan multiple sessions.
- Seek help when needed. (Our job is to help you!)
- Use the Piazza online forum for Q&A about assignments, course material.

What’s Ahead?

Lecture 2. Intro to Java.

Precept 1. Meets today/tomorrow.

Not registered? Go to any precept now; officially register ASAP.

Change precepts? Use SCORE.

Assignment 0. [www.princeton.edu/~cos126/assignments.php]
- Due Monday 9 PM.
- 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
Goods and Bads of One-Time Pads

**Good.**
- Easily computed by hand.
- Very simple encryption/decryption processes.
- Provably unbreakable if bits are truly random. [Shannon, 1940s]

**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.

"one time" means one time only

"impractical for Web commerce"

Pseudo-Random Bit Generator

**Practical middle-ground.**
- Make a "random" bit generator gadget.
- Alice and Bob each get identical small gadgets.
- Also, matching initial values, or "seeds," for their gadgets

**Goal.** Small gadget that produces a long sequence of bits.

Small deterministic gadgets that produce long sequences of pseudo-random bits:
- Enigma
- Linear feedback shift register.
- Linear congruential generator.
- Blum-Blum-Shub generator.
- [many others have been invented]

Pseudo-random? Bits are not really random:
- Bob's and Alice's gadgets must produce the same bits from the same seed.
- Bits must have as many properties of random bits as possible (to foil Eve).

Ex 1. approx 1/2 0s and 1/2 1s
Ex 2. approx 1/4 each of 00, 01, 10, 11

"Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin."
— John von Neumann (left)
— ENIAC (right)

Shift Register

**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 Russian one-time pad"
If not, what is the pattern?

\{(8, 10)\} linear feedback shift register.

- Shift register with 11 cells.
- Bit \(b_8\) is XOR of previous bits \(b_6\) and \(b_{10}\).
- Pseudo-random bit = \(b_0\).

\begin{align*}
\text{Linear Feedback Shift Register (LFSR)}
\end{align*}

\begin{align*}
\text{Linear Feedback Shift Register Demo}
\end{align*}

\begin{align*}
\text{Random Numbers}
\end{align*}

Q. Are these 2000 numbers random?

If not, what is the pattern?

\begin{align*}
A. \text{No. This is output of \{(8, 10)\} LFSR with seed 01101000010!}
\end{align*}

\begin{align*}
\text{LFSR Encryption}
\end{align*}

Encryption.

- Convert text message to \(N\) bits.
- Initialize LFSR with given seed
- Generate \(N\) bits with LFSR.
- Take bitwise XOR of two bitstrings.
- Convert binary back to text.
LFSR Decryption

Decryption.
- Convert encrypted message to binary.
- Initialize identical LFSR with same seed.
- Generate N bits with LFSR.
- Take bitwise XOR of two bitstrings.
- Convert back into text.

Key properties of LFSRs

Property 1: A zero fill (all 0s) produces all 0s.
- Don’t use all 0s as a seed!
- Fill of all 0s will not otherwise occur.

Property 2: Bitstream must eventually cycle.
- $2^N-1$ nonzero fills in an N-bit register.
- Future output completely determined by current fill.

Property 3: Cycle length in an N-bit register is at most $2^N-1$.
- Could be smaller; cycle length depends on tap positions.
- Need higher math (theory of finite groups) to know tap positions for given N.

Bottom line: 11-bit register generates at most 2047 bits before cycling, so use a longer register (say, N = 61).

Goods and Bads of LFSRs

Good.
- Easily computed with simple machine.
- Very simple encryption/decryption processes.
- Bits have many of the same properties as random bits.
- Scalable: 20 cells for 1 million bits; 30 cells for 1 billion bits.
  [but need theory of finite groups to know where to put taps]

Bad.
- Still need secure, independent way to distribute LFSR seed.
- The bits are not truly random.
  [bits in our 11-bit LFSR cycle after $2^{11} - 1 = 2047$ steps]
- Experts have cracked LFSR encryption.
  [need more complicated machines]
Other LFSR Applications

What else can we do with a LFSR?
- DVD encryption with CSS.
- DVD decryption with DeCSS!
- Subroutine in military cryptosystems.

DVD Jon
(Norwegian hacker)

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 seed[];
    private final int tap;
    private final int N;
    public LFSR(String seed, 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.print(lfsr.step());
    }
}
```