## CS 101 Today...

### Jotto Corner

<table>
<thead>
<tr>
<th>5C guess</th>
<th>ZD guess</th>
<th>HS guess</th>
<th>ZD guess</th>
</tr>
</thead>
<tbody>
<tr>
<td>party: 3</td>
<td>diner: 1</td>
<td>alien: 2</td>
<td>diner: 2</td>
</tr>
<tr>
<td>??????: ?</td>
<td>savvy: ?</td>
<td>??????: 0</td>
<td>savvy: ?</td>
</tr>
</tbody>
</table>

Santiago

Jeff

### Looking Back

- Computing as composition
  - *clay == functions*

### Looking Forward

- Computing as representation
  - *clay == data & bits*

---

Our top-10 list of binary jokes...

- On a scale of 1 to 10, how likely is it that this question is using binary?
- What's a 4?
- ...4?

- There are only 10 types of people in the world: Those who understand binary and those who don't.
Some legs to stand on...?

This is heady stuff!

Creating more and more capable compositions

Rotating $c$ by $n$ leads to encipher and decipher functions, with ord and chr as their arguments.
Some legs to stand on!

It looks like I'm ahead of this...

creating more and more capable compositions

how are even these fundamentals physically realized?!

It looks like I'm ahead of this...
Binary Storage & Representation

The SAME bits can represent different pieces of data, depending on type:

- **str**: 'C'
- **int**: \( \chi = 42 \)

The same bits are in each container:

- **C**: 00101010
- **\( \chi \)**: 00101010

8 bits = 1 byte = 1 box

But why these bits?
What is 42?
What is 42?

42

4 tens + 2 ones

100 + 20 + 3

1 hundred + 2 tens + 3 ones
Value (semantics)
stuff we care about (what things mean)

Syntax
stuff we need to communicate
Value (semantics)
stuff we care about (what things *mean*)

Syntax
stuff we need *to communicate*
Write 123 in binary...

Base 2

$101010_2$

Base 10

$42_{10}$

- 4 tens + 2 ones

$123_{10}$

- 1 hundred + 2 tens + 3 ones

Each column represents the base's next power.
Write 123 in binary...

Base 2:

\[
\begin{array}{cccccccc}
\text{EIGHTS} & \text{FOURS} & \text{TWOs} & \text{ONES} \\
2^7 & 2^6 & 2^5 & 2^4 \\
& & 25 & 24 \\
\end{array}
\]

Base 10:

\[
\begin{aligned}
123 &= 1(64) + 1(32) + 1(16) + 0(8) + 1(4) + 1(2) + 1(1) \\
&= 1 \text{ hundred } + 2 \text{ tens } + 3 \text{ ones}
\end{aligned}
\]
### Binary math

<table>
<thead>
<tr>
<th>+</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

### Decimal math

<table>
<thead>
<tr>
<th>+</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
</tr>
</tbody>
</table>

#### Tables of basic facts

### Addition

#### Multiplication

[www.youtube.com/watch?v=Nh7xapVB-Wk](www.youtube.com/watch?v=Nh7xapVB-Wk)
Convert these two binary numbers to decimal:

\[
\begin{array}{cccccc}
32 & 16 & 8 & 4 & 2 & 1 \\
11 & 0 & 0 & 1 & 1 & 1 \\
\end{array}
\]

\[32 + 16 + 2 + 1 = 51\]

\[
\begin{array}{cccccc}
32 & 16 & 8 & 4 & 2 & 1 \\
1 & 0 & 0 & 0 & 1 & 0 \\
\end{array}
\]

\[8 + 128 = 136\]

Convert these two decimal numbers to binary:

\[
\begin{array}{cccccc}
32 & 16 & 8 & 4 & 2 & 1 \\
0 & 1 & 1 & 0 & 0 & 0 \\
\end{array}
\]

\[28_{10}\]

\[
\begin{array}{cccccc}
32 & 16 & 8 & 4 & 2 & 1 \\
1 & 0 & 1 & 1 & 0 & 1 \\
\end{array}
\]

\[101_{10}\]

Add these two binary numbers:

\[101101 + 1110 \rightarrow 111011\]

Multiply these binary numbers:

\[101101 \times 1110 = \overline{1000000000} \overline{1011100} \overline{01100100} \overline{1011010010} \]

Extra! Can you figure out the last binary digit (bit) of 53 without determining any other bits? The last two? 3?
Convert these two binary numbers *to decimal*:

![Binary to Decimal Conversion](image)

Convert these two decimal numbers *to binary*:

![Decimal to Binary Conversion](image)

**Extra!** Can you figure out the last binary digit (bit) of 53 *without determining any other bits?* The last two? 3?

We'll return to this *in a bit...*
Add these two binary numbers *WITHOUT* converting to decimal!

\[
\begin{array}{cccccc}
32 & 16 & 8 & 4 & 2 & 1 \\
529 & & & & & \\
\hline
+ & 742 & & & & \\
\hline
1271 & & & & & \\
\end{array}
\]

Hint:

Do you remember this algorithm? It's the same!

\[
\begin{array}{cccccc}
101101 & & & & & \\
+ & 1110 & & & & \\
\hline
111111 & & & & & \\
\end{array}
\]
Add these two binary numbers *WITHOUT* converting to decimal!

\[
\begin{array}{cccccc}
32 & 16 & 8 & 4 & 2 & 1 \\
\hline
101101 & & & & & \\
+ & 1110 & & & & \\
\hline
& & & & & \\
\end{array}
\]

\[
\begin{array}{cccc}
& & & \\
& & 529 & \\
+ & 742 & \\
\hline
& 1271 & \\
\end{array}
\]

Hint:
Do you remember this algorithm? It's the same!
Multiply these two binary numbers **WITHOUT** converting to decimal!

```
011101
*
1110
```

**Goal**

```
529
*  42
1058
+ 2116
22218
```

**Hint:**

Do you remember this algorithm? It's the same!

A machine could - and probably **should** - be doing this!
Multiply these two binary numbers **WITHOUT** converting to decimal!

```
  101101
* 1110
```

```
  000000
1011010
10110100
+ 101101000
```

```
10011110110
```

**Hint:**
Do you remember this algorithm? It’s the same!

```
529
* 42
1058
+ 2116
22218
```

"partial products"

A machine could **and probably should** - be doing this!
There are 10 kinds of "people" in the universe:
those who know ternary,
those who don't, and
those who think this is a binary joke!
Which of these \textit{isn't} 42...?

and what are the bases of the rest?

\begin{align*}
\text{base 2} & \quad 101010 \\
\text{base 3} & \quad 1120 \\
\text{base 4} & \\
\text{base 5} & \\
\text{base 6} & \\
\text{base 7} & \\
\text{base 8} & \\
\text{base 9} & \\
\text{base 10} & \\
\text{base 11} & \\
\text{base 12} & \\
\text{base 16} & \\
\end{align*}

\begin{align*}
\text{digits: } & 0, 1 \\
\text{digits: } & 0, 1, 2 \\
\text{digits: } & 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 \\
\text{digits: } & 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, X \\
\end{align*}

Beyond Binary
All 42s!
base 1                           

base 2    ——— 101010

base 3    ——— 1120

base 16    ——— 2A

digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

digits: 0, 1

digits: 0, 1, 2

digits: 1
Off base?

"Duodecimal Society"
"Dozenal Society"

Base 12 –

Base 20: Americas

Olmec base-20 numbers
E. Mexico, ~ 300 AD

Base 27: New Guinea

Base 60 – Ancient Sumeria

Telefol is a language spoken by the Telefol people in Papua New Guinea, notable for possessing a base-27 numeral system.

Some of these bases are still echoing around...
But *why* binary?
Ten symbols is too many!

A computer has to differentiate physically among all its possibilities.

ten symbols ~ ten different voltages

This is too difficult to replicate billions of times

What digits are these?

Ouch!
Ten symbols is too many!

A computer has to differentiate physically among all its possibilities.

Ten symbols ~ ten different voltages

This is too difficult to replicate billions of times

What digits are these?

Ouch!
Two symbols is easiest!

A computer has to differentiate **physically** among all its possibilities.

Ten symbols ~ ten different voltages

two symbols ~ two different voltages

What digits are these?
Two symbols is easiest!

A computer has to differentiate physically among all its possibilities.

ten symbols ~ ten different voltages

two symbols ~ two different voltages

What digits are these?
Ternary computers?

50 of these *Setun* ternary machines were made at Moscow U. ~ **1958**

This project was discontinued in 1970... *though not because of the ternary design!*
Eye-catching submissions...

ASCII wanderings...

Noah H.

Jack W.
Kristen M.

James K.

Elizabeth S.

Marco H.

Trees...
Spirals...

Jack W.

Lindsey L.

Kripesh R.

Michael L.
Flakes/Petals...
Well-named images!

Rainbow tri-blade boomerang.jpg

Violet and blue-studded snowflake kite.jpg
a turtle-drawn portrait from turtle graphics...
Back to bits…

not the original name…
b.d. ~ binary digit ~ bit

"bit" first appeared in print in 1948 (Claude Shannon)

Extra! Can you figure out the **last binary digit** (bit) of **53** without determining any earlier bits? The last **two**? **three**? **All** of them?
Lab 5: Computing in binary

This first step of **left-to-right** conversion into binary is tricky to program... *Why?*

You mean *aside* from having to think in binary?
**Lab 5: Computing in binary**

<table>
<thead>
<tr>
<th>base 2</th>
<th>base 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>128 64 32 16 8 4 2 1</td>
<td>100 10 1</td>
</tr>
</tbody>
</table>

= 141

This first step of **left-to-right** conversion into binary is tricky to program... *Why?*

It's tricky to find the largest power needed...
in the end, we need "53"-worth of value

Extra! Can you figure out the **last binary digit** (bit) of 53 without determining any earlier bits? The last **two**? **three**? **All** of them?
Lab 5: Converting to binary...

What does the fact that 141 is ODD tell us?!

Try right-to-left!

141 = 10001101
Lab 5: Computing in binary

You'll write these right! (right-to-left)

<table>
<thead>
<tr>
<th>base 10</th>
<th>base 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>100 10 1</td>
<td>128 64 32 16 8 4 2 1</td>
</tr>
<tr>
<td>141</td>
<td>'10001101'</td>
</tr>
</tbody>
</table>

Right-to-left works!

numToBinary( N )

we need to represent binary numbers with strings

binaryToNum( S )

n2b(141)

b2n('10001101')
Lab 5: Computing in binary

```
def numToBinary(N):
    if N == 0:
        return ''
    elif N%2 == 0:
        return numToBinary(N) + '0'
    else:
        return numToBinary(N) + '1'
```

If N is even, what is the final bit?
If N is odd, what is the final bit?

base 10

<table>
<thead>
<tr>
<th>100</th>
<th>10</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>141</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

base 2

<table>
<thead>
<tr>
<th>128</th>
<th>64</th>
<th>32</th>
<th>16</th>
<th>8</th>
<th>4</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

How much VALUE is left to convert!?
Lab 5: Binary conversion

```python
def numToBinary( N ):
    if N == 0: return ''
    elif N%2 == 0:
        return numToBinary(N//2) + '0'
    else:
        return numToBinary(N//2) + '1'
```

- If N is even, what is the final bit?
- If N is odd, what is the final bit?
- How much VALUE is left to convert??
Reasoning, *bit by bit*

![Image of reasoning process involving bitwise operations and shift operators.](image-url)
Reasoning, *bit by bit*

**left-shift**

left-shift by 1

11  
110

3 << 1  
6

left-shift by 2

11  
1100

3 << 2  
12

right-shift**

right-shift by 1

101010  
10101

42 >> 1  
21

42 >> 2  
?

What does *left-shifting* do to the value of a #?

What does *right-shifting* do to the value of a #?
Being bit-wise

In processors shifts, ands, ors, adds, and subtractions are very fast, whereas multiplying, dividing, and mod, which are relatively slow.

Given this, what is a way to compute these expressions using only fast operations, maybe in combination?

\[
\begin{align*}
\text{N} \div 4 \\
\text{N} \times 7 \\
\text{N} \times 17 \\
\text{N} \mod 16
\end{align*}
\]
In processors shifts, ands, ors, adds, and subtractions are *very fast*, whereas multiplying, dividing, and mod, which are relatively *slow*.

Given this, what is a way to compute these expressions using *only fast* operations, maybe in combination?

\[
\begin{align*}
    N \div 4 \\
    N \times 7 \\
    N \times 17 \\
    N \mod 16
\end{align*}
\]

Let's first look at *why* you'd bother ... !?
In processors **shift**, **and**, **or**, **add**, and **subtract** are **much faster** than **multiply**, **divide**, and **mod**, which are **relatively slow**.

Old Microsoft *systems-interview* question, #42:

42. Give a fast way to multiply a number by 7.

42. How would you check if a word is a palindrome?
Intel x86 processor instructions and their speeds (2014)

In processors **shift**, **and**, **or**, **add**, and **subtract** are **much faster** than **multiply**, **divide**, and **mod**, which are **relatively slow**.

Given this, what is a way to compute these statements using combinations from only the **fast** operations above?

\[
\begin{align*}
N \mod 4 &\iff N \gg \_ \\
N \times 7 &\iff N \\
N \times 17 &\iff N \\
N \% 16 &\iff N
\end{align*}
\]
In processors **shift**, and, or, add, and **subtract** are **much faster** than multiply, divide, and **mod**, which are **relatively slow**.

Given this, what is a way to compute these statements using combinations from only the **fast** operations above?

\[
\begin{align*}
N \mathbin{\complement}/4 & \iff N \gg 2 \\
N \cdot 7 & \iff (N \ll 3) - N \\
N \cdot 17 & \iff (N \ll 4) + N \\
N \% 16 & \iff N - (N \gg 4) \ll 4
\end{align*}
\]