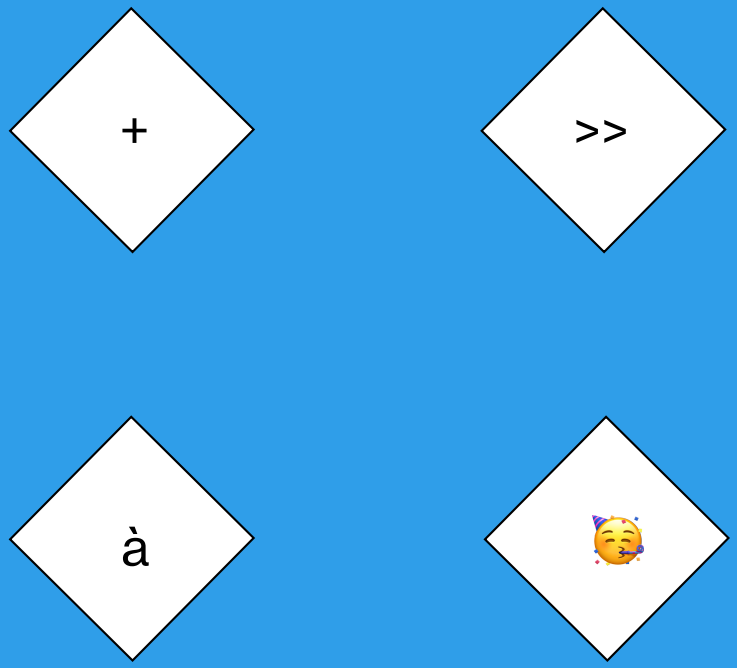


Computer Systems

Binary Arithmetic in Python

CS51 – Spring 2026

Administrative

Assignment 2

Assignment 3 available (complete first part before lab)

Lab tomorrow

Mentor hour attendance


There are 10 types
of people in the world
Those who understand binary
and those who don't

Bitwise operations

Bitwise operations

Treat numbers as twos complement numbers

Perform Boolean operation **per bit** of two numbers

& - AND

X	Y	X ∧ Y
0	0	0
0	1	0
1	0	0
1	1	1

~ - NOT

X	¬ X
0	1
1	0

| - OR

X	Y	X ∨ Y
0	0	0
0	1	1
1	0	1
1	1	1

^ - XOR

X	Y	X ⊕ Y
0	0	0
0	1	1
1	0	1
1	1	0

Bitwise operations: & (bitwise-AND)

6 & 5 =

AND

X	Y	X ∧ Y
0	0	0
0	1	0
1	0	0
1	1	1

Convert numbers to binary (twos complement)

For each digit, computer the Boolean operation

Convert resulting Boolean number back to an int

Bitwise operations: & (bitwise-AND)

6 & 5 =

AND

X	Y	X ∧ Y
0	0	0
0	1	0
1	0	0
1	1	1

Convert numbers to binary (twos complement)

For each digit, compute the Boolean operation

Convert resulting Boolean number back to an int

Bitwise operations: & (bitwise-AND)

$$6 \;\&\; 5 = \begin{matrix} 0\,1\,1\,0_2 \\ 0\,1\,0\,1_2 \end{matrix}$$

AND

X	Y	X ∧ Y
0	0	0
0	1	0
1	0	0
1	1	1

Convert numbers to binary (twos complement)

For each digit, compute the Boolean operation

Convert resulting Boolean number back to an int

Bitwise operations: & (bitwise-AND)

$$6 \,\&\, 5 = \begin{array}{r} 0\,1\,1\,0_2 \\ \&\; 0\,1\,0\,1_2 \\ \hline \end{array}$$

AND

X	Y	X ∧ Y
0	0	0
0	1	0
1	0	0
1	1	1

Convert numbers to binary (twos complement)

For each digit, compute the Boolean operation

Convert resulting Boolean number back to an int

Bitwise operations: & (bitwise-AND)

$$6 \,\&\, 5 = \begin{array}{r} 0\,1\,1\,0_2 \\ \&\; 0\,1\,0\,1_2 \\ \hline 0\,1\,0\,0_2 \end{array}$$

AND

X	Y	X ∧ Y
0	0	0
0	1	0
1	0	0
1	1	1

Convert numbers to binary (twos complement)

For each digit, compute the Boolean operation

Convert resulting Boolean number back to an int

Bitwise operations: & (bitwise-AND)

$$6 \;\&\; 5 = \begin{array}{r} 0\,1\,1\,0_2 \\ \&\; 0\,1\,0\,1_2 \\ \hline 0\,1\,0\,0_2 \end{array}$$

AND

X	Y	X ∧ Y
0	0	0
0	1	0
1	0	0
1	1	1

Convert numbers to binary (twos complement)

For each digit, compute the Boolean operation

Convert resulting Boolean number back to an int

Bitwise operations: & (bitwise-AND)

$$6 \,\&\, 5 = \begin{array}{r} 0\,1\,1\,0_2 \\ \&\; 0\,1\,0\,1_2 \\ \hline 0\,1\,0\,0_2 \end{array} = 4$$

AND

X	Y	X ∧ Y
0	0	0
0	1	0
1	0	0
1	1	1

Convert numbers to binary (twos complement)

For each digit, compute the Boolean operation

Convert resulting Boolean number back to an int

Bitwise operations: | (bitwise-OR)

11 | -9 =

OR

X	Y	X v Y
0	0	0
0	1	1
1	0	1
1	1	1

Convert numbers to binary (twos complement)

For each digit, compute the Boolean operation

Convert resulting Boolean number back to an int

Bitwise operations: | (bitwise-OR)

$11 \mid -9 =$

$0\,1\,0\,1\,1_2$

$1\,0\,1\,1\,1_2$

OR

X	Y	X v Y
0	0	0
0	1	1
1	0	1
1	1	1

Convert numbers to binary (twos complement)

For each digit, compute the Boolean operation

Convert resulting Boolean number back to an int

Bitwise operations: | (bitwise-OR)

$$11 \mid -9 = \begin{array}{r} 0\,1\,0\,1\,1_2 \\ \mid\ 1\,0\,1\,1\,1_2 \\ \hline \end{array}$$

OR

X	Y	X v Y
0	0	0
0	1	1
1	0	1
1	1	1

Convert numbers to binary (twos complement)

For each digit, compute the Boolean operation

Convert resulting Boolean number back to an int

Bitwise operations: | (bitwise-OR)

$$
\begin{array}{rr}
11 \mid -9 = & 0\,1\,0\,1\,1_2 \\
 & \mid\ 1\,0\,1\,1\,1_2 \\
\hline
 & 1\,1\,1\,1\,1_2
\end{array}
$$

OR

X	Y	X v Y
0	0	0
0	1	1
1	0	1
1	1	1

Convert numbers to binary (twos complement)

For each digit, compute the Boolean operation

Convert resulting Boolean number back to an int

Bitwise operations: | (bitwise-OR)

$$11 \mid -9 = \begin{array}{r} 0\,1\,0\,1\,1_2 \\ \mid\ 1\,0\,1\,1\,1_2 \\ \hline 1\,1\,1\,1\,1_2 \end{array}$$

OR

X	Y	X v Y
0	0	0
0	1	1
1	0	1
1	1	1

Convert numbers to binary (twos complement)

For each digit, compute the Boolean operation

Convert resulting Boolean number back to an int

Bitwise operations: | (bitwise-OR)

$$11 \mid -9 = \begin{array}{r} 0\,1\,0\,1\,1_2 \\ \mid\ 1\,0\,1\,1\,1_2 \\ \hline 1\,1\,1\,1\,1_2 \end{array} = -1$$

OR

X	Y	X v Y
0	0	0
0	1	1
1	0	1
1	1	1

Convert numbers to binary (twos complement)

For each digit, compute the Boolean operation

Convert resulting Boolean number back to an int

Bitwise operations: ~ (bitwise-NOT)

~9 =

NOT

X	¬ X
0	1
1	0

Convert numbers to binary (twos complement)

For each digit, compute the Boolean operation

Convert resulting Boolean number back to an int

Bitwise operations: ~ (bitwise-NOT)

$\sim 9 = 01001_2$

NOT

X	¬ X
0	1
1	0

For positive numbers, make sure to include a leading 0

Convert numbers to binary (twos complement)

For each digit, compute the Boolean operation

Convert resulting Boolean number back to an int

Bitwise operations: ~ (bitwise-NOT)

$\sim 9 = 01001_2$

NOT

X	¬ X
0	1
1	0

Convert numbers to binary (twos complement)

For each digit, compute the Boolean operation

Convert resulting Boolean number back to an int

Bitwise operations: ~ (bitwise-NOT)

$$\sim 9 = 01001_2 = 10110_2$$

NOT

X	¬ X
0	1
1	0

Convert numbers to binary (twos complement)

For each digit, compute the Boolean operation

Convert resulting Boolean number back to an int

Bitwise operations: ~ (bitwise-NOT)

$$\sim 9 = 01001_2 = 10110_2$$

NOT

X	¬ X
0	1
1	0

Convert numbers to binary (twos complement)

For each digit, compute the Boolean operation

Convert resulting Boolean number back to an int

Bitwise operations: ~ (bitwise-NOT)

$$\sim 9 = 01001_2 = 10110_2 = -10$$

NOT

X	¬ X
0	1
1	0

Convert numbers to binary (twos complement)

For each digit, compute the Boolean operation

Convert resulting Boolean number back to an int

Bitwise operations: length differences

10 ^ 1 =

^ - XOR

X	Y	X ⊕ Y
0	0	0
0	1	1
1	0	1
1	1	0

Convert numbers to binary (twos complement)

For each digit, compute the Boolean operation

Convert resulting Boolean number back to an int

Bitwise operations: length differences

$10 \text{ ^ } 1 = \quad 0\,1\,0\,1\,0_2$

$0\,1_2$

^ - XOR

X	Y	X ⊕ Y
0	0	0
0	1	1
1	0	1
1	1	0

Convert numbers to binary (twos complement)

For each digit, compute the Boolean operation

Convert resulting Boolean number back to an int

Bitwise operations: length differences

$10 \text{ ^ } 1 =$

$$\begin{array}{r} 0\,1\,0\,1\,0_2 \\ \text{^} \qquad 0\,1_2 \\ \hline \end{array}$$

What do we do here?

^ - XOR

X	Y	X ⊕ Y
0	0	0
0	1	1
1	0	1
1	1	0

Convert numbers to binary (twos complement)

For each digit, compute the Boolean operation

Convert resulting Boolean number back to an int

Bitwise operations: length differences

$10 \text{ ^ } 1 =$

$$\begin{array}{r} 01010_2 \\ \text{^ } \quad 00001_2 \\ \hline \end{array}$$

^ - XOR

X	Y	X ⊕ Y
0	0	0
0	1	1
1	0	1
1	1	0

Fill with zeros

Convert numbers to binary (twos complement)

For each digit, compute the Boolean operation

Convert resulting Boolean number back to an int

Bitwise operations: length differences

$$10 \text{ ^ } 1 = \begin{array}{r} 01010_2 \\ \text{^ } \quad 00001_2 \\ \hline 01011_2 \end{array}$$

^ - XOR

X	Y	X ⊕ Y
0	0	0
0	1	1
1	0	1
1	1	0

Convert numbers to binary (twos complement)

For each digit, compute the Boolean operation

Convert resulting Boolean number back to an int

Bitwise operations: length differences

$$10 \text{ ^ } 1 = \begin{array}{r} 01010_2 \\ \text{^ } \quad 00001_2 \\ \hline 01011_2 \end{array}$$

^ - XOR

X	Y	X ⊕ Y
0	0	0
0	1	1
1	0	1
1	1	0

Convert numbers to binary (twos complement)

For each digit, compute the Boolean operation

Convert resulting Boolean number back to an int

Bitwise operations: length differences

$$10 \text{ ^ } 1 = \begin{array}{r} 0\,1\,0\,1\,0_2 \\ \text{^}\ \ 0\,0\,0\,0\,1_2 \\ \hline 0\,1\,0\,1\,1_2 \end{array} = 11$$

^ - XOR

X	Y	X ⊕ Y
0	0	0
0	1	1
1	0	1
1	1	0

Convert numbers to binary (twos complement)

For each digit, compute the Boolean operation

Convert resulting Boolean number back to an int

An aside

What is -3 in twos complement using:

- Using 3 bits:
- Using 4 bits:
- Using 5 bits:
- Using 32 bits:

An aside: -3

Using 3 bits

$$1\ 0\ 1$$

$$-2^2\ 2^1\ 1^0$$

$$-4\ 2\ 1$$

An aside: -3

Using 4 bits

$$1\ 1\ 0\ 1$$

$$-2^3\ 2^2\ 2^1\ 1^0$$

$$-8\ \ 4\ \ 2\ \ 1$$

An aside: -3

Using 5 bits

$$1\ 1\ 1\ 0\ 1$$

$$-2^4\ 2^3\ 2^2\ 2^1\ 1^0$$

$$-16\ 8\ 4\ 2\ 1$$

An aside: -3

1 1 0 1

$-2^3\ 2^2\ 2^1\ 1^0$

-8 4 2 1

1 1 1 0 1

$-2^4\ 2^3\ 2^2\ 2^1\ 1^0$

-16 8 4 2 1

When we add a bit, we subtract 16

An aside: -3

1 1 0 1

-2^3 2^2 2^1 1^0

-8 4 2 1

1 1 1 0 1

-2^4 2^3 2^2 2^1 1^0

-16 8 4 2 1

When we add a bit, we subtract 16

But we also add 16 (-8 becomes 8)

An aside

number of bits	-3
3	101
4	1101
5	11101
6	111101
7	1111101
8	11111101
9	111111101
10	1111111101
...	1.....101

An aside: -3

Using 4 bits

$$1\ 1\ 0\ 1$$

$$-2^2\ 2^1\ 1^0$$

$$-4\ 2\ 1$$

Bitwise operations: length differences

10 ^ -1 = $0\,1\,0\,1\,0_2$

^ - XOR

X	Y	X ⊕ Y
0	0	0
0	1	1
1	0	1
1	1	0

What do we do for -1?

Convert numbers to binary (twos complement)

For each digit, compute the Boolean operation

Convert resulting Boolean number back to an int

Bitwise operations: length differences

10 ^ -1 = $0\,1\,0\,1\,0_2$

$1\,1\,1\,1\,1_2$

^ - XOR

X	Y	X ⊕ Y
0	0	0
0	1	1
1	0	1
1	1	0

Convert numbers to binary (twos complement)

For each digit, compute the Boolean operation

Convert resulting Boolean number back to an int

Bitwise operations: length differences

$10 \text{ ^ } -1 =$

$$\begin{array}{r} 0\,1\,0\,1\,0_2 \\ 1\,1\,1\,1\,1_2 \\ \hline \end{array}$$

^ - XOR

X	Y	X ⊕ Y
0	0	0
0	1	1
1	0	1
1	1	0

Convert numbers to binary (twos complement)

For each digit, compute the Boolean operation

Convert resulting Boolean number back to an int

Bitwise operations: length differences

$$10 \text{ ^ } -1 = \begin{array}{r} 0\,1\,0\,1\,0_2 \\ \text{^}\ \ 1\,1\,1\,1\,1_2 \\ \hline 1\,0\,1\,0\,1_2 \end{array}$$

^ - XOR

X	Y	X ⊕ Y
0	0	0
0	1	1
1	0	1
1	1	0

Convert numbers to binary (twos complement)

For each digit, compute the Boolean operation

Convert resulting Boolean number back to an int

Bitwise operations: length differences

$$10 \text{ ^ } -1 = \begin{array}{r} 0\,1\,0\,1\,0_2 \\ \text{^}\ \ 1\,1\,1\,1\,1_2 \\ \hline 1\,0\,1\,0\,1_2 \end{array}$$

^ - XOR

X	Y	X ⊕ Y
0	0	0
0	1	1
1	0	1
1	1	0

Convert numbers to binary (twos complement)

For each digit, compute the Boolean operation

Convert resulting Boolean number back to an int

Bitwise operations: length differences

$$10 \text{ ^ } -1 = \begin{array}{r} 0\,1\,0\,1\,0_2 \\ \text{^}\ \ 1\,1\,1\,1\,1_2 \\ \hline 1\,0\,1\,0\,1_2 \end{array} = -11$$

^ - XOR

X	Y	X ⊕ Y
0	0	0
0	1	1
1	0	1
1	1	0

Convert numbers to binary (twos complement)

For each digit, compute the Boolean operation

Convert resulting Boolean number back to an int

PRACTICE TIME – Bitwise operations

$x = 5_{10}$ and $y = -9_{10}$

- $\sim x$
- $\sim y$
- $x \& y$
- $x | y$
- $x \wedge y$

& - AND

X	Y	X ∧ Y
0	0	0
0	1	0
1	0	0
1	1	1

~ - NOT

X	¬ X
0	1
1	0

| - OR

X	Y	X ∨ Y
0	0	0
0	1	1
1	0	1
1	1	1

^ - XOR

X	Y	X ⊕ Y
0	0	0
0	1	1
1	0	1
1	1	0

ANSWER – Bitwise operations

$x = 5_{10}$ and $y = -9_{10}$

- $\sim x = 11010_2 = -6_{10}$
- $\sim y = 01000_2 = 8_{10}$
- $x \& y = 00101_2 = 5_{10}$
- $x | y = 10111_2 = -9_{10}$
- $x \wedge y = 10010_2 = -14_{10}$

& - AND

X	Y	X ∧ Y
0	0	0
0	1	0
1	0	0
1	1	1

~ - NOT

X	¬ X
0	1
1	0

| - OR

X	Y	X ∨ Y
0	0	0
0	1	1
1	0	1
1	1	1

^ - XOR

X	Y	X ⊕ Y
0	0	0
0	1	1
1	0	1
1	1	0

Masking

We can use bitwise operations to **mask** a certain number of bits in a number.

Masking resets, sets, or inverts certain bits

Bitwise AND will reset a subset of the bits to 0 (or can also be seen as copying a subset of bits and zeroing everything else out)

Bitwise OR will set a subset of the bits to 1

Bitwise XOR will invert a subset of the bits

Bitwise AND – Resetting bits to 0

We want to reset the three least significant bits to 0

	Bit	6^{th}	5^{th}	4^{th}	3^{rd}	2^{nd}	1^{st}	0^{th}
	Data	0/1	0/1	0/1	0/1	0/1	0/1	0/1
	Mask							
&	**Result**							

What number should we bitwise-AND (&) to do this?

Bitwise AND – Resetting bits to 0

We want to reset the three least significant bits to 0

	Bit	6th	5th	4th	3rd	2nd	1st	0th
	Data	0/1	0/1	0/1	0/1	0/1	0/1	0/1
	Mask	1	1	1	1	0	0	0
&	**Result**							

What number should we bitwise-AND (&) to do this?

Bitwise AND – Resetting bits to 0

We want to reset the three least significant bits to 0

	Bit	6^{th}	5^{th}	4^{th}	3^{rd}	2^{nd}	1^{st}	0^{th}
	Data	0/1	0/1	0/1	0/1	0/1	0/1	0/1
	Mask	1	1	1	1	0	0	0
&	**Result**							

-8 4 2 1

-8

Bitwise AND – Resetting bits to 0

For example, $47_{10} = 0101111_2$ we want to reset the three least significant bits to 0

	Bit	6^{th}	5^{th}	4^{th}	3^{rd}	2^{nd}	1^{st}	0^{th}
	Data	0	1	0	1	1	1	1
	Mask	1	1	1	1	0	0	0
&	**Result**							

& the mask

Bitwise AND – Resetting bits to 0

For $47_{10} = 0101111_2$ we want to reset the three least significant bits to 0

	Bit	6^{th}	5^{th}	4^{th}	3^{rd}	2^{nd}	1^{st}	0^{th}
	Data	0	1	0	1	1	1	1
	Mask	1	1	1	1	0	0	0
&	**Result**	0	1	0	1	**0**	**0**	**0**

& the mask

Bitwise AND – Resetting bits to 0

For $47_{10} = 0101111_2$ we want to reset the three least significant bits to 0

	Bit	6^{th}	5^{th}	4^{th}	3^{rd}	2^{nd}	1^{st}	0^{th}
	Data	0	1	0	1	1	1	1
	Mask	1	1	1	1	0	0	0
&	**Result**	0	1	0	1	**0**	**0**	**0**

40

PRACTICE TIME - Bitwise AND

What would the result be of applying the mask 42_{10} on the input 85_{10} using bitwise AND?.

ANSWER - Bitwise AND

- What would the result be of applying the mask 42_{10} on the input 85_{10} using bitwise AND?
- First, we convert from decimal to binary, working with 8 bits:
  - $42_{10} = 00101010_2$
  - $85_{10} = 01010101_2$
- We then apply the bitwise AND operation which will reset the the 7th (no real effect), 6th, 4th, 2nd, and 0th bit of the input, resulting to 0_{10}:

Bit	7th	6th	5th	4th	3rd	2nd	1st	0th
Data	0	1	0	1	0	1	0	1
Mask	0	0	1	0	1	0	1	0
Result	**0**	**0**	0	**0**	0	**0**	0	**0**

Bitwise OR – Setting bits to 1

We want to set the first two bits to 1

Bit	7^{th}	6^{th}	5^{th}	4^{th}	3^{rd}	2^{nd}	1^{st}	0^{th}
Data	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Mask								
Result								

What number should we bitwise-OR (|) to do this?

Bitwise OR – Setting bits to 1

We want to set the first two bits to 1

Bit	7^{th}	6^{th}	5^{th}	4^{th}	3^{rd}	2^{nd}	1^{st}	0^{th}
Data	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Mask	1	1	0	0	0	0	0	0
Result								
	-128	64	32	16	8	4	2	1

What number should we bitwise-OR (|) to do this?

Bitwise OR – Setting bits to 1

We want to set the first two bits to 1

Bit	7th	6th	5th	4th	3rd	2nd	1st	0th
Data	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Mask	1	1	0	0	0	0	0	0
Result								
	-128	64	32	16	8	4	2	1

-64

Bitwise OR – Setting bits to 1

For example, $47_{10} = 00101111$, we want to set the first two bits to 1

Bit	7th	6th	5th	4th	3rd	2nd	1st	0th
Data	0	0	1	0	1	1	1	1
Mask	1	1	0	0	0	0	0	0
Result	**1**	**1**	1	0	1	1	1	1
	-128	64	32	16	8	4	2	1

Bitwise OR – Setting bits to 1

For example, $47_{10} = 00101111$, we want to set the first two bits to 1

Bit	7th	6th	5th	4th	3rd	2nd	1st	0th
Data	0	0	1	0	1	1	1	1
Mask	1	1	0	0	0	0	0	0
Result	**1**	**1**	1	0	1	1	1	1
	-128	64	32	16	8	4	2	1

-17

PRACTICE TIME - Bitwise OR

What would the result be of applying the mask 17_{10} on the input -27_{10} using bitwise OR?

ANSWER - Bitwise OR

- What would the result be of applying the mask 17_{10} on the input -27_{10} using bitwise OR? You can assume you have 8 bits.
- First, we convert from decimal to binary, working with 8 bits:
  - $17_{10} = 010001_2$
  - $-27_{10} = 100101_2$
- We then apply the bitwise OR operation which will set the the 4th and 0th bit of the input to 1, resulting to -11_{10}:

Bit	5th	4th	3rd	2nd	1st	0th
Data	1	0	0	1	0	1
Mask	0	1	0	0	0	1
Result	1	**1**	0	1	0	**1**

Bitwise XOR – Inverting bits

We want to invert the first two and last two digits and leave the four in middle unchanged

	Bit	7^{th}	6^{th}	5^{th}	4^{th}	3^{rd}	2^{nd}	1^{st}	0^{th}
	Data	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
	Mask								
^	**Result**								

What number should we bitwise-XOR (^) to do this?

Bitwise XOR – Inverting bits

We want to invert the first two and last two digits and leave the four in middle unchanged

	Bit	7^{th}	6^{th}	5^{th}	4^{th}	3^{rd}	2^{nd}	1^{st}	0^{th}
	Data	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
	Mask	1	1	0	0	0	0	1	1
^	**Result**								
		-128	64	32	16	8	4	2	1

What number should we bitwise-XOR (^) to do this?

Bitwise XOR – Inverting bits

We want to invert the first two and last two digits and leave the four in middle unchanged

	Bit	7^{th}	6^{th}	5^{th}	4^{th}	3^{rd}	2^{nd}	1^{st}	0^{th}
	Data	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
	Mask	1	1	0	0	0	0	1	1
^	**Result**								
		-128	64	32	16	8	4	2	1

-61

Bitwise XOR – Inverting bits

For example, $117_{10} = 01110101,$ we want to invert the first two and last two digits and leave the four in middle unchanged

	Bit	7th	6th	5th	4th	3rd	2nd	1st	0th
	Data	0	1	1	1	0	1	0	1
	Mask	1	1	0	0	0	0	1	1
^	**Result**	**1**	**0**	1	1	0	1	**1**	**0**
		-128	64	32	16	8	4	2	1

Bitwise XOR – Inverting bits

For example, $117_{10} = 01110101,$ we want to invert the first two and last two digits and leave the four in middle unchanged

	Bit	7th	6th	5th	4th	3rd	2nd	1st	0th
	Data	0	1	1	1	0	1	0	1
	Mask	1	1	0	0	0	0	1	1
^	**Result**	**1**	**0**	1	1	0	1	**1**	**0**
		-128	64	32	16	8	4	2	1

-74

PRACTICE TIME - Mystery 1

What does the following Python function do?

```
def mystery1(a, b):
      return (a ^ b) == 0
```

ANSWER - Mystery 1

What does the following Python function do?

```
def mystery1(a, b):
        return (a ^ b) == 0
```

- The function takes two numbers (a,b), applies the bitwise XOR (a^b), turns it into a decimal and compares the result to 0. If it is equal to 0, it returns True, otherwise it returns False.
- If both numbers have n bits, we have $a_{n-1}a_{n-2} \dots a_0$
- Tests if a and b are equal.
- $5_{10} \wedge 5_{10}$ is $0101_2 \wedge 0101_2 = 0000_2 = 0_{10}$
- $5_{10} \wedge 6_{10}$ is $0101_2 \wedge 0110_2 = 0100_2 = 4_{10}$

PRACTICE TIME - Mystery 2

What does the following Python function do? You can assume we are working with 4 bits

```
def mystery2(a):
      return (a & (1<<3)) != 0
```

ANSWER - Mystery 2

What does the following Python function do? You can assume we are working with 32 bits

```
def mystery2(a):
        return (a & (1<<3)) != 0
```

- Checks whether a is negative by isolating the MSB.

PRACTICE TIME - Mystery 3

What does the following Python function do?

```
def mystery3(a,b):
      return (a & (1<<b)) != 0
```

ANSWER - Mystery 3

What does the following Python function do?

```
def mystery3(a,b):
      return (a & (1<<b)) != 0
```

Tests if the b-th bit in a is 1.

PRACTICE TIME - Mystery 4

What does the following Python function do?

```
def mystery4(a):
      return a | 1
```

ANSWER - Mystery 4

- What does the following Python function do?

```
def mystery4(a):
      return a | 1
```

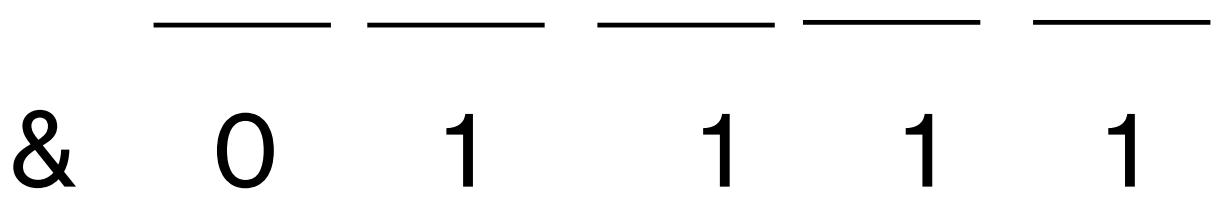
If a is even, it returns a+1, if a is odd, it leaves it unchanged.

bit_length()

Write a function that removes the most significant bit of a positive number

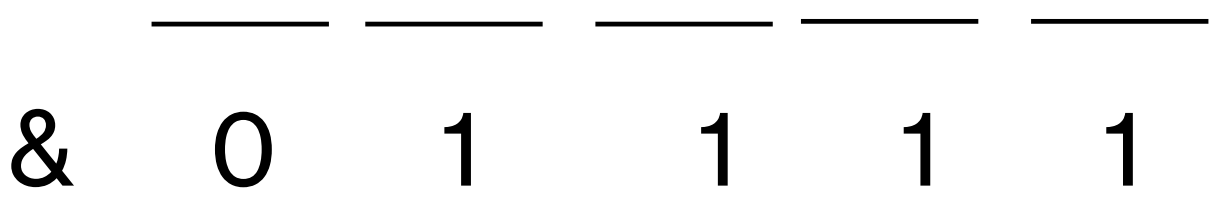
bit_length()

Write a function that removes the most significant bit of a positive number



bit_length()

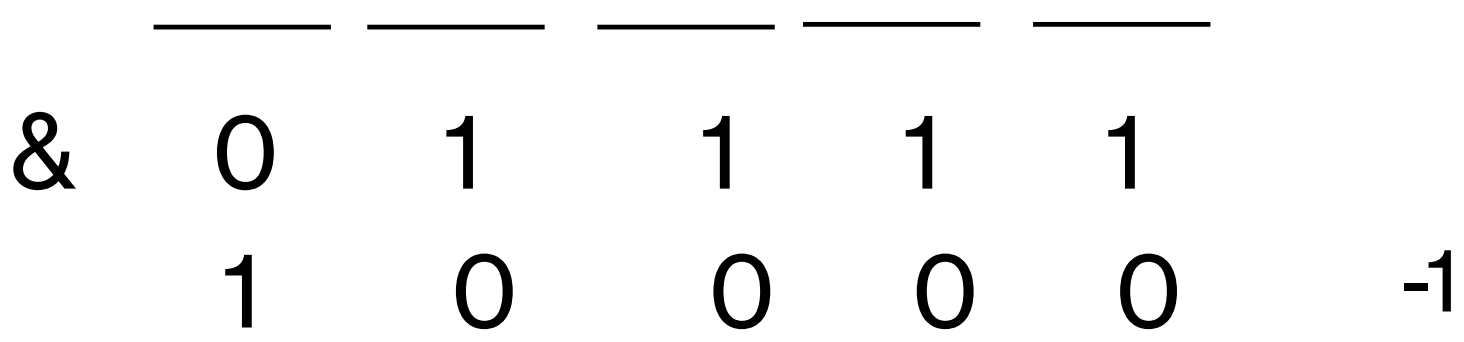
Write a function that removes the most significant bit of a number



How can we get this number?

bit_length()

Write a function that removes the most significant bit of a number



bit_length()

Returns the number of bits in a number (ignoring the sign bit)

bit_length()

Write a function that removes the most significant bit of a positive number

```
def remove_most_significant_bit(a):
       bits = a.bit_length()
       mask = (1 << bits - 1) - 1
       return a & mask
```

Why bitwise operators?

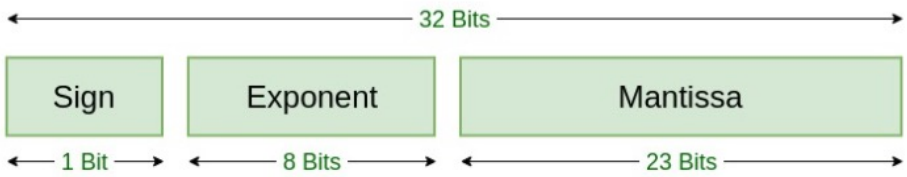
Represent many Boolean values with a single integer (e.g., in memory constrained environments)

Are fast, so can speed up computation (if you can post your problem as a bitwise operation)

Interacting with low-level hardware

Beyond numbers

Floating point numbers



Single Precision
IEEE 754 Floating-Point Standard

Double Precision
IEEE 754 Floating-Point Standard

Representing information

- But what about non-integer numbers, such as floating-point numbers, or any other type of information, such as letters, emojis, colors, sound etc?

- There are different international standards that determine how binary is **encoded** into other representations.

- For example, the most used standard to encode real numbers is the IEEE 754 standard.

Text

Dec	Oct	Hex	C	Dec	Oct	Hex	C	Dec	Oct	Hex	C	Dec	Oct	Hex	C
0	0	0	^@	32	40	20		64	100	40	@	96	140	60	`
1	1	1	^A	33	41	21	!	65	101	41	A	97	141	61	a
2	2	2	^B	34	42	22	"	66	102	42	B	98	142	62	b
3	3	3	^C	35	43	23	#	67	103	43	C	99	143	63	c
4	4	4	^D	36	44	24	$	68	104	44	D	100	144	64	d
5	5	5	^E	37	45	25	%	69	105	45	E	101	145	65	e
6	6	6	^F	38	46	26	&	70	106	46	F	102	146	66	f
7	7	7	^G	39	47	27	'	71	107	47	G	103	147	67	g
8	10	8	^H	40	50	28	(	72	110	48	H	104	150	68	h
9	11	9	^I	41	51	29	)	73	111	49	I	105	151	69	i
10	12	a	^J	42	52	2a	*	74	112	4a	J	106	152	6a	j
11	13	b	^K	43	53	2b	+	75	113	4b	K	107	153	6b	k
12	14	c	^L	44	54	2c	,	76	114	4c	L	108	154	6c	l
13	15	d	^M	45	55	2d	-	77	115	4d	M	109	155	6d	m
14	16	e	^N	46	56	2e	.	78	116	4e	N	110	156	6e	n
15	17	f	^O	47	57	2f	/	79	117	4f	O	111	157	6f	o
16	20	10	^P	48	60	30	0	80	120	50	P	112	160	70	p
17	21	11	^Q	49	61	31	1	81	121	51	Q	113	161	71	q
18	22	12	^R	50	62	32	2	82	122	52	R	114	162	72	r
19	23	13	^S	51	63	33	3	83	123	53	S	115	163	73	s
20	24	14	^T	52	64	34	4	84	124	54	T	116	164	74	t
21	25	15	^U	53	65	35	5	85	125	55	U	117	165	75	u
22	26	16	^V	54	66	36	6	86	126	56	V	118	166	76	v
23	27	17	^W	55	67	37	7	87	127	57	W	119	167	77	w
24	30	18	^X	56	70	38	8	88	130	58	X	120	170	78	x
25	31	19	^Y	57	71	39	9	89	131	59	Y	121	171	79	y
26	32	1a	^Z	58	72	3a	:	90	132	5a	Z	122	172	7a	z
27	33	1b	^[	59	73	3b	;	91	133	5b	[	123	173	7b	{
28	34	1c	^\	60	74	3c	<	92	134	5c	\	124	174	7c	|
29	35	1d	^]	61	75	3d	=	93	135	5d	]	125	175	7d	}
30	36	1e	^^	62	76	3e	>	94	136	5e	^	126	176	7e	~
31	37	1f	^_	63	77	3f	?	95	137	5f	_	127	177	7f	

Encoding text - ASCII

ASCII (American Standard Code for Information Interchange) which was established in the ‘60s and used 7 bits to represent 128 characters: These included the capital and lowercase letters of the English alphabet, digits 0-9, punctuation and special symbols like @.

For example, 01000011 01010011 00110101 00110001 00100001 represents the text “CS51!”

Encoding text - Unicode

- In 1990s, **Unicode** expanded encoding to thousands of characters to account for all different schemes individual languages used and even includes emojis!
- The most common implementations of Unicode are **UTF-8** ,which was designed for backward compatibility with ASCII, and **UTF-16**.

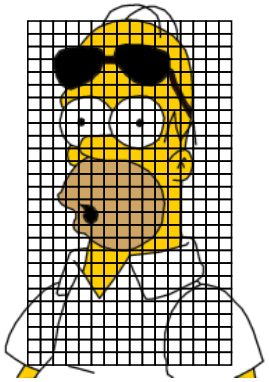
Smileys & Emotion

face-smiling

№	Code	Browser	Sample	GMail	SB	DCM	KDDI	CLDR Short Name
1	U+1F600				—	—	—	grinning face
2	U+1F603							grinning face with big eyes
3	U+1F604					—	—	grinning face with smiling eyes
4	U+1F601							beaming face with smiling eyes
5	U+1F606				—		—	grinning squinting face
6	U+1F605				—		—	grinning face with sweat
7	U+1F923			—	—	—	—	rolling on the floor laughing
8	U+1F602					—		face with tears of joy
9	U+1F642				—	—	—	slightly smiling face
10	U+1F643			—	—	—	—	upside-down face

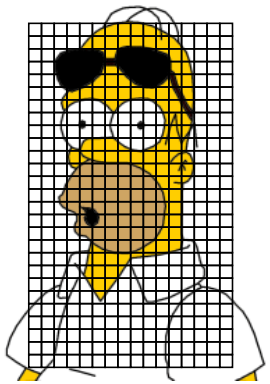
How is an image represented?

How is an image represented?



- images are made up of pixels
- for a color image, each pixel corresponds to an RGB value (i.e. three numbers)

Image features



for each pixel: R[0-255]
G[0-255]
B[0-255]

Encoding images

- Images consist of pixels, the smallest image unit.
- Each pixel stores a **color** using three **color channels**: Red, Green, Blue (RGB).
- Each color channel can be encoded using binary numbers
  - Typical: 8 bits per channel, that is 24 bits per pixel. With 8 bits, we can represent 0,..., 255
  - For example, 00000000 represents no red and 11111111 represents full red. Same for green, blue.
- Putting it together, we have triples of numbers to represent pixels. E.g.,
  - Pure red = $11111111\ 00000000\ 00000000 \rightarrow$ (255,0,0) – only red
  - Pure white = $11111111\ 11111111\ 11111111 \rightarrow$ (255,255,255) – all three colors
  - Pure black = $00000000\ 00000000\ 00000000 \rightarrow$ (0, 0, 0) – absence of all three colors
- An entire image would be represented as a matrix of pixels (width × height), each pixel's RGB values encoded in binary

PRACTICE TIME - Encoding images

f we have 8 bits for each of the three color channels, how many colors can we support in a 24-bit RGB system?

ANSWER - Encoding images

- If we have 8 bits for each of the three color channels, how many colors can we support in a 24-bit RGB system?
- Each channel supports $2^8 = 256$ possible values.
- Thus, the total number of supported colors is $256 \times 256 \times 256 = 16{,}777{,}216$
- That means that a 24-bit RGB system, can represent over 16 million colors.

RGBA format

- Sometimes, the RGB format is supplemented with one more channel called the **alpha channel**.
- The alpha channel indicates how opaque a pixel is.
- By convention RGBA colors are stored in hex. For example, for alpha, 00 would be fully transparent and FF fully opaque. For colors, 00 would be lack of color and FF would be pure color.
- For example, the RGBA color #FF00FF80 is a semi-transparent purple:
  - FF_{16} stands for a full red in the red channel
  - 00_{16} stands for no green in the green channel
  - FF_{16} stands for a full blue in the blue channel
  - and $80_{16} = 128_{10}$ represents 50% opacity.

Practice Problems – Problem 1

- Convert the following decimal numbers to their *unsigned* binary representation and add them:
  - $45_{10} + 27_{10}$
  - How many bits do you need to not have overflow?
- Assume 6-bit signed numbers in two's complement representation. Add them and state whether overflow occurs:
  - $101110_2 + 010101_2$
  - Convert these three numbers to decimal to double check your work.
- Add the hex numbers 0xA7+0x5C
  - Convert these three numbers to decimal to double check your work.

Practice Problems – Answer 1

- Convert the following decimal numbers to their *unsigned* binary representation and add them:
  - $45_{10} + 27_{10} = 101101_2 + 011011_2 = 1001000$
  - How many bits do you need to not have overflow? 7
- Assume 6-bit signed numbers in two's complement representation. Add them and state whether overflow occurs:
  - $101110_2 + 010101_2 = 000011_2$. No overflow
  - Convert these three numbers to decimal to double check your work. -18+21=-3
- Add the hex numbers $A7_{16} + 5C_{16}$
  - 103_{16}
  - Convert these three numbers to decimal to double check your work. $167_{10} + 92_{10} = 259_{10}$

Practice Problems – Problem 2

- What will the results of the following operations be in binary and decimal assuming 8 bits?

- $-28_{10} \gg 2$

Practice Problems – Answer 2

- What will the results of the following operations be in binary and decimal assuming 8 bits?

- $-28_{10} \gg 2 \; = \; 11100100_2 \gg 2 \; = 11111001_2 = -7_{10}$

Practice Problems – Problem 3

- Compute the following expressions in Python:
- $101101012_2 \& 11110000_2$
- $01011011_2 | 00101101_2$
- $11001010_2 \wedge 10101100_2$

Practice Problems – Answer 3

- Compute the following expressions in Python:
- $101101012_2 \& 11110000_2 = 10110000_2$
- $01011011_2 | 00101101_2 = 01111111_2$
- $11001010_2 \wedge 10101100_2 = 01100110_2$