

# One Bit At A Time

- We can combine bits, like with base-10 numbers, to represent more data.  
**8 bits = 1 byte.**
- Computer memory is just a large array of bytes! It is *byte-addressable*; you can't address (store location of) a bit; only a byte.
- Computers still fundamentally operate on bits; we have just gotten more creative about how to represent different data as bits!
  - Images
  - Audio
  - Video
  - Text
  - And more...

# How does a bit do so much?

- Information can be reshaped
- Numbers can have the same value but in different representations
- Typically, we use base 10 in everyday life (most people attribute this to humans having 10 fingers, but humans have used other # systems)
- Base 10 has ten digits: 0 1 2 3 4 5 6 7 8 9
- Base 2 has two digits: 0 1
- We can represent up to ten numbers with one digit in base 10
- We can represent up to two numbers with one digit in base 2
- If we want to represent more numbers, we add more digits regardless of the base.

# Base 10

5 9 3 4

Digits 0-9 (*base-10*)

4 Columns

# Base 10

5 9 3 4

↑   ↑   ↑   ↑

thousands   hundreds   tens   ones

$$= 5 \cdot 1000 + 9 \cdot 100 + 3 \cdot 10 + 4 \cdot 1$$

# Base 10

5 9 3 4

↑   ↑   ↑   ↑

$10^3$   $10^2$   $10^1$   $10^0$

$$= 5 \cdot 10^3 + 9 \cdot 10^2 + 3 \cdot 10^1 + 4 \cdot 10^0$$

# Base 10

5 9 3 4

$10^x$ :

3

2

1

0

# Base 2

$2^x$ :      1   0   1   1  
             3   2   1   0

Digits 0-1 (*base-2*)

# Base 2

1 0 1 1  
 $2^3$   $2^2$   $2^1$   $2^0$

$$= 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 11_{10}$$



# Base 2

Most significant bit (MSB)

Least significant bit (LSB)

1 0 1 1  
eights fours twos ones

$$= 1*8 + 0*4 + 1*2 + 1*1 = 11_{10}$$

# Base 10 to Base 2

**Question:** What is 6 in base 2?

- 2 Strategies:
  1. Build the number from the left (Find the most significant bit first)
  2. Build the number from the right (Find the least significant bit first)

# Base 10 to Base 2: Most Significant Bit First

**Question:** What is 6 in base 2?

- Strategy:
  - What is the largest power of 2  $\leq 6$ ?  $2^2=4$

0	1		
<hr/>	<hr/>	<hr/>	<hr/>
$2^3$	$2^2$	$2^1$	$2^0$

# Base 10 to Base 2: Most Significant Bit First

**Question:** What is 6 in base 2?

- Strategy:
  - What is the largest power of 2  $\leq 6$ ?  $2^2=4$

0	1		
<hr/>	<hr/>	<hr/>	<hr/>
$2^3$	$2^2$	$2^1$	$2^0$

# Base 10 to Base 2: Most Significant Bit First

**Question:** What is 6 in base 2?

- Strategy:
  - What is the largest power of 2  $\leq 6$ ?  $2^2=4$
  - Now, what is the largest power of 2  $\leq 6 - 2^2$ ?

0	1		
<hr/>	<hr/>	<hr/>	<hr/>
$2^3$	$2^2$	$2^1$	$2^0$

# Base 10 to Base 2: Most Significant Bit First

**Question:** What is 6 in base 2?

- Strategy:

- What is the largest power of 2  $\leq 6$ ?  $2^2=4$
- Now, what is the largest power of 2  $\leq 6 - 2^2$ ?  $2^1=2$

0	1	1	
—	—	—	—
$2^3$	$2^2$	$2^1$	$2^0$

# Base 10 to Base 2: Most Significant Bit First

**Question:** What is 6 in base 2?

• Strategy:

- What is the largest power of 2  $\leq 6$ ?  $2^2=4$
- Now, what is the largest power of 2  $\leq 6 - 2^2$ ?  $2^1=2$
- $6 - 2^2 - 2^1 = 0$ !

$$\begin{array}{cccc} 0 & 1 & 1 & \\ \hline 2^3 & 2^2 & 2^1 & 2^0 \end{array}$$

# Base 10 to Base 2: Most Significant Bit First

**Question:** What is 6 in base 2?

• Strategy:

- What is the largest power of 2  $\leq 6$ ?  $2^2=4$
- Now, what is the largest power of 2  $\leq 6 - 2^2$ ?  $2^1=2$
- $6 - 2^2 - 2^1 = 0$ !

$$\begin{array}{cccc} 0 & 1 & 1 & 0 \\ \hline 2^3 & 2^2 & 2^1 & 2^0 \end{array}$$



# Base 10 to Base 2: Most Significant Bit First

**Question:** What is 6 in base 2?

• Strategy:

- What is the largest power of 2  $\leq 6$ ?  $2^2=4$
- Now, what is the largest power of 2  $\leq 6 - 2^2$ ?  $2^1=2$
- $6 - 2^2 - 2^1 = 0$ !

$$\begin{array}{cccc} 0 & 1 & 1 & 0 \\ \hline 2^3 & 2^2 & 2^1 & 2^0 \\ \hline \end{array}$$
$$= 0*8 + 1*4 + 1*2 + 0*1 = 6$$

# Base 10 to Base 2: Least Significant Bit First

**Question:** What is 6 in base 2?

- Strategy:
  - What is the largest power of 2  $\leq 6$ ?

# Base 10 to Base 2: Least Significant Bit First

## Question: What is 6 in base 2?

- Strategy:

- What is  $6 \% 2$ ? **0** – Use the remainder as the value for the bit
- What is  $6 // 2$ ? **3** – Use the integer quotient as the starting value for the next operations

$$\begin{array}{r} 0 \\ \hline 2^3 \quad 2^2 \quad 2^1 \quad 2^0 \end{array}$$

# Base 10 to Base 2: Least Significant Bit First

**Question:** What is 6 in base 2?

- Strategy:

- What is  $6 \% 2$ ? **0**
- What is  $6 // 2$ ? **3**

$$\begin{array}{cccc} & & 1 & 0 \\ \hline & & 2^1 & 2^0 \\ \hline \end{array}$$

# Base 10 to Base 2: Least Significant Bit First

**Question:** What is 6 in base 2?

- Strategy:

- What is  $1 \% 2$ ? **1**
- What is  $1 // 2$ ? **0** – Stop when the Integer Divide returns 0

$$\begin{array}{cccc} & 1 & 1 & 0 \\ \hline & 2^2 & 2^1 & 2^0 \end{array}$$

# Base 10 to Base 2: Least Significant Bit First

**Question:** What is 6 in base 2?

- Strategy:
  - Add Leading Zeroes as needed

$$\begin{array}{cccc} 0 & 1 & 1 & 0 \\ \hline 2^3 & 2^2 & 2^1 & 2^0 \end{array}$$

## Practice: Base 2 to Base 10

What is the base-2 value 1010 in base-10?

- a) 20
- b) 101
- c) 10
- d) 5
- e) Other

## Practice: Base 10 to Base 2

What is the base-10 value 14 in base 2?

- a) **1111**
- b) **1110**
- c) **1010**
- d) **Other**



# Byte Values

- What is the minimum and maximum base-10 value a single byte (8 bits) can store?
- Please answer minimum first

# Byte Values

- What is the minimum and maximum base-10 value a single byte (8 bits) can store?      **minimum = 0**      **maximum = ?**

# Byte Values

- What is the minimum and maximum base-10 value a single byte (8 bits) can store?      **minimum = 0**      **maximum = ?**

$2^x$ :      1 1 1 1 1 1 1 1  
             7 6 5 4 3 2 1 0

# Byte Values

- What is the minimum and maximum base-10 value a single byte (8 bits) can store?      **minimum = 0**      **maximum = ?**

2<sup>x</sup>:      7   6   5   4   3   2   1   0

1 1 1 1 1 1 1 1

- Strategy 1:**  $1 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 255$

# Byte Values

- What is the minimum and maximum base-10 value a single byte (8 bits) can store?      **minimum = 0**      **maximum = 255**

2<sup>x</sup>:      7   6   5   4   3   2   1   0

1 1 1 1 1 1 1 1

- **Strategy 1:**  $1*2^7 + 1*2^6 + 1*2^5 + 1*2^4 + 1*2^3 + 1*2^2 + 1*2^1 + 1*2^0 = 255$
- **Strategy 2:**  $2^8 - 1 = 255$

# Byte Values

- How about minimum and maximum base-10 value for 16 bits?  
**minimum = 0      maximum = ?**

# Combinations of bits can Encode Anything

We can encode anything we want with bits. E.g., the ASCII character set.

## ASCII Code: Character to Binary

0	0011 0000	O	0100 1111	m	0110 1101
1	0011 0001	P	0101 0000	n	0110 1110
2	0011 0010	Q	0101 0001	o	0110 1111
3	0011 0011	R	0101 0010	p	0111 0000
4	0011 0100	S	0101 0011	q	0111 0001
5	0011 0101	T	0101 0100	r	0111 0010
6	0011 0110	U	0101 0101	s	0111 0011
7	0011 0111	V	0101 0110	t	0111 0100
8	0011 1000	W	0101 0111	u	0111 0101
9	0011 1001	X	0101 1000	v	0111 0110
A	0100 0001	Y	0101 1001	w	0111 0111
B	0100 0010	Z	0101 1010	x	0111 1000
C	0100 0011	a	0110 0001	y	0111 1001
D	0100 0100	b	0110 0010	z	0111 1010
E	0100 0101	c	0110 0011	.	0010 1110
F	0100 0110	d	0110 0100	,	0010 0111
G	0100 0111	e	0110 0101	:	0011 1010
H	0100 1000	f	0110 0110	;	0011 1011
I	0100 1001	g	0110 0111	?	0011 1111
J	0100 1010	h	0110 1000	!	0010 0001
K	0100 1011	I	0110 1001	'	0010 1100
L	0100 1100	j	0110 1010	"	0010 0010
M	0100 1101	k	0110 1011	{	0010 1000
N	0100 1110	l	0110 1100	}	0010 1001
				space	0010 0000

# CS107: Three Number Representations

**Unsigned Integers:** positive integers and zero only

Ex. 0, 1, 2, ..., 74629, 99999999

**Signed Integers:** negative, positive, and zero integers only

Ex. 0, 1, 2, ..., 74629, 99999999

(represented in "two's complement")

**Floating Point Numbers:** a base-2 representation of scientific notation, for real numbers

Ex. 0.0, 0.1, -12.2,  $4.87563 \times 10^3$ ,  $-1.00005 \times 10^{-12}$



# Number Representations

- **Unsigned Integers:** positive and 0 integers. (e.g. 0, 1, 2, ... 99999...)
- **Signed Integers:** negative, positive and 0 integers. (e.g. ...-2, -1, 0, 1,... 9999...)
- **Floating Point Numbers:** real numbers. (e.g. 0.1, -12.2,  $1.5 \times 10^{12}$ )
  - ↳ Look up IEEE floating point if you're interested! Or wait till week 7 😊 !

# Data Sizes

On the myth computers (and most 64-bit computers today), the `int` representation is comprised of 32-bits, or four 8-bit bytes. NOTE: C language does not mandate sizes. To the right is Figure 2.3 from your textbook:

C declaration		Bytes	
Signed	Unsigned	32-bit	64-bit
[signed] char	unsigned char	1	1
short	unsigned short	2	2
int	unsigned	4	4
long	unsigned long	4	8
int32_t	uint32_t	4	4
int64_t	uint64_t	8	8
char *		4	8
float		4	4
double		8	8

# Data Sizes

There are guarantees on the lower-bounds for type sizes, but you should expect that the myth machines will have the numbers in the 64-bit column.

C declaration		Bytes	
Signed	Unsigned	32-bit	64-bit
[signed] char	unsigned char	1	1
short	unsigned short	2	2
int	unsigned	4	4
long	unsigned long	4	8
int32_t	uint32_t	4	4
int64_t	uint64_t	8	8
char *		4	8
float		4	4
double		8	8

# Data Sizes

You can be guaranteed the sizes  
for `int32_t` (4 bytes) and  
`int64_t` (8 bytes)

C declaration		Bytes	
Signed	Unsigned	32-bit	64-bit
[signed] char	unsigned char	1	1
short	unsigned short	2	2
int	unsigned	4	4
long	unsigned long	4	8
int32_t	uint32_t	4	4
int64_t	uint64_t	8	8
char *		4	8
float		4	4
double		8	8

# Data Sizes

C allows a variety of ways to order keywords to define a type. The following all have the same meaning:

```
unsigned long
unsigned long int
long unsigned
long unsigned int
```

C declaration		Bytes	
Signed	Unsigned	32-bit	64-bit
[signed] char	unsigned char	1	1
short	unsigned short	2	2
int	unsigned	4	4
long	unsigned long	4	8
int32_t	uint32_t	4	4
int64_t	uint64_t	8	8
char *		4	8
float		4	4
double		8	8

# Transitioning To Larger Datatypes



- **Early 2000s:** most computers were **32-bit**. This means that pointers were **4 bytes (32 bits)**.
- 32-bit pointers store a memory address from 0 to  $2^{32}-1$ , equaling  **$2^{32}$  bytes of addressable memory**. This equals **4 Gigabytes**, meaning that 32-bit computers could have at most **4GB** of memory (RAM)!
- Because of this, computers transitioned to **64-bit**. This means that datatypes were enlarged; pointers in programs were now **64 bits**.
- 64-bit pointers store a memory address from 0 to  $2^{64}-1$ , equaling  **$2^{64}$  bytes of addressable memory**. This equals **16 Exabytes**, meaning that 64-bit computers could have at most  **$1024*1024*1024*16$  GB** of memory (RAM)!

# Addressing and Byte Ordering



On the myth machines, pointers are 64-bits long, meaning that a program can "address" up to  $2^{64}$  bytes of memory, because each byte is individually addressable.

This is a lot of memory! It is 16 exabytes, or  $1.84 \times 10^{19}$  bytes. Older, 32-bit machines could only address  $2^{32}$  bytes, or 4 Gigabytes.

64-bit machines can address 4 *billion* times more memory than 32-bit machines...

Machines will not need to address more than  $2^{64}$  bytes of memory for a long, long

# Byte Range

Because a byte is made up of 8 bits, we can represent the range of a byte as follows:

00000000 to 11111111

This range is 0 to 255 in decimal.

But, neither binary nor decimal is particularly convenient to write out bytes (binary is too long, and decimal isn't numerically friendly for byte representation)

So, we use "hexadecimal," (base 16).



# Hexadecimal

- When working with bits, oftentimes we have large numbers with 32 or 64 bits.
- Instead, we'll represent bits in *base-16 instead*; this is called **hexadecimal**.

0110 1010 0011

0-15 0-15 0-15

# Hexadecimal

- Hexadecimal is *base-16*, so we need digits for 1-15. How do we do this?

0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
										10	11	12	13	14	15

# Hexadecimal

Hexadecimal has 16 digits, so we augment our normal 0-9 digits with six more digits: A, B, C, D, E, and F.

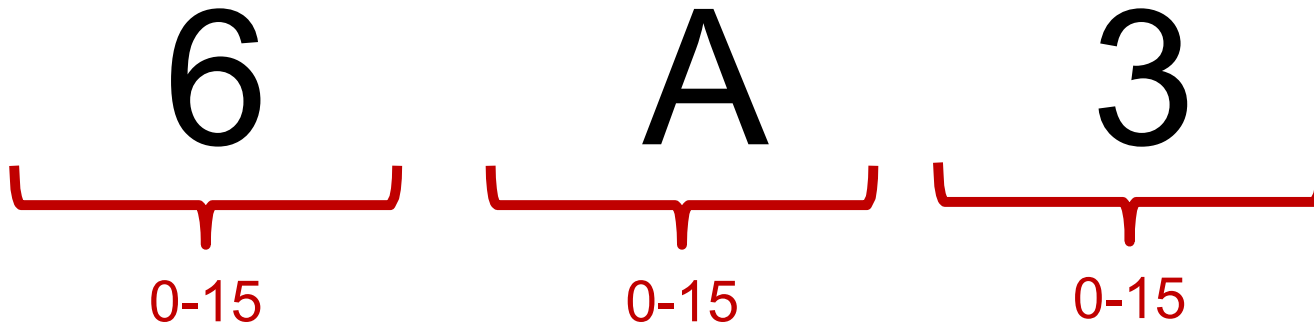
Figure 2.2 in the textbook shows the hex digits and their binary and decimal values:

Hex digit	0	1	2	3	4	5	6	7
Decimal value	0	1	2	3	4	5	6	7
Binary value	0000	0001	0010	0011	0100	0101	0110	0111

Hex digit	8	9	A	B	C	D	E	F
Decimal value	8	9	10	11	12	13	14	15
Binary value	1000	1001	1010	1011	1100	1101	1110	1111

# Hexadecimal

- When working with bits, oftentimes we have large numbers with 32 or 64 bits.
- Instead, we'll represent bits in *base-16 instead*; this is called **hexadecimal**.



Each is a base-16 digit!

# Hexadecimal

- We distinguish hexadecimal numbers by prefixing them with **0x**, and binary numbers with **0b**. These prefixes also work in C
- E.g. **0xf5** is **0b11110101**

0x f 5

1111 0101

# Practice: Hexadecimal to Binary

What is **0x173A** in binary?

Hexadecimal	1	7	3	A
Binary	0001	0111	0011	1010

# Practice: Hexadecimal to Binary

What is **0b1111001010** in hexadecimal? (*Hint: start from the right*)

Binary	11	1100	1010
Hexadecimal	3	C	A

# Hexadecimal

Convert: 0b1111001010110110110011 to hexadecimal.

Binary	11	1100	1010	1101	1011	0011
Hexadecimal	3	C	A	D	B	3

(start from the **right**)

0b1111001010110110110011  
is hexadecimal 3CADB3

Hex digit	0	1	2	3	4	5	6	7
Decimal value	0	1	2	3	4	5	6	7
Binary value	0000	0001	0010	0011	0100	0101	0110	0111

Hex digit	8	9	A	B	C	D	E	F
Decimal value	8	9	10	11	12	13	14	15
Binary value	1000	1001	1010	1011	1100	1101	1110	1111



# Hexadecimal

Convert: 0b1111001010110110110011 to hexadecimal.

Binary	11	1100	1010	1101	1011	0011
Hexadecimal	3	C	A	D	B	3

(start from the **right**)

0b1111001010110110110011  
is hexadecimal 3CADB3

Hex digit	0	1	2	3	4	5	6	7
Decimal value	0	1	2	3	4	5	6	7
Binary value	0000	0001	0010	0011	0100	0101	0110	0111

Hex digit	8	9	A	B	C	D	E	F
Decimal value	8	9	10	11	12	13	14	15
Binary value	1000	1001	1010	1011	1100	1101	1110	1111

# Hexadecimal

Convert: 0b1111001010110110110011 to hexadecimal.

Binary	11	1100	1010	1101	1011	0011
Hexadecimal	3	C	A	D	B	3

(start from the **right**)

0b1111001010110110110011  
is hexadecimal 3CADB3

Hex digit	0	1	2	3	4	5	6	7
Decimal value	0	1	2	3	4	5	6	7
Binary value	0000	0001	0010	0011	0100	0101	0110	0111

Hex digit	8	9	A	B	C	D	E	F
Decimal value	8	9	10	11	12	13	14	15
Binary value	1000	1001	1010	1011	1100	1101	1110	1111

# Hexadecimal

Convert: 0b1111001010110110110011 to hexadecimal.

Binary	11	1100	1010	1101	1011	0011
Hexadecimal	3	C	A	D	B	3

(start from the **right**)

0b1111001010110110110011  
is hexadecimal 3CADB3

Hex digit	0	1	2	3	4	5	6	7
Decimal value	0	1	2	3	4	5	6	7
Binary value	0000	0001	0010	0011	0100	0101	0110	0111

Hex digit	8	9	A	B	C	D	E	F
Decimal value	8	9	10	11	12	13	14	15
Binary value	1000	1001	1010	1011	1100	1101	1110	1111

# Hexadecimal

Convert: 0b1111001010110110110011 to hexadecimal.

Binary	11	1100	1010	1101	1011	0011
Hexadecimal	3	C	A	D	B	3

(start from the **right**)

0b1111001010110110110011  
is hexadecimal 3CADB3

Hex digit	0	1	2	3	4	5	6	7
Decimal value	0	1	2	3	4	5	6	7
Binary value	0000	0001	0010	0011	0100	0101	0110	0111

Hex digit	8	9	A	B	C	D	E	F
Decimal value	8	9	10	11	12	13	14	15
Binary value	1000	1001	1010	1011	1100	1101	1110	1111

# Hexadecimal

Convert: 0b1111001010110110110011 to hexadecimal.

Binary	11	1100	1010	1101	1011	0011
Hexadecimal	3	C	A	D	B	3

(start from the **right**)

0b1111001010110110110011  
is hexadecimal 3CADB3

Hex digit	0	1	2	3	4	5	6	7
Decimal value	0	1	2	3	4	5	6	7
Binary value	0000	0001	0010	0011	0100	0101	0110	0111

Hex digit	8	9	A	B	C	D	E	F
Decimal value	8	9	10	11	12	13	14	15
Binary value	1000	1001	1010	1011	1100	1101	1110	1111

# Hexadecimal

Convert: 0b1111001010110110110011 to hexadecimal.

Binary	11	1100	1010	1101	1011	0011
Hexadecimal	3	C	A	D	B	3

(start from the **right**)

0b1111001010110110110011  
is hexadecimal 3CADB3

Hex digit	0	1	2	3	4	5	6	7
Decimal value	0	1	2	3	4	5	6	7
Binary value	0000	0001	0010	0011	0100	0101	0110	0111

Hex digit	8	9	A	B	C	D	E	F
Decimal value	8	9	10	11	12	13	14	15
Binary value	1000	1001	1010	1011	1100	1101	1110	1111

# Decimal to Hexadecimal

To convert from decimal to hexadecimal, you need to repeatedly divide the number in question by 16, and the remainders make up the digits of the hex number:

314156 decimal:

314,156 / 16	= 19,634 with 12 remainder:	C
19,634 / 16	= 1,227 with 2 remainder:	2
1,227 / 16	= 76 with 11 remainder:	B
76 / 16	= 4 with 12 remainder:	C
4 / 16	= 0 with 4 remainder:	4

Reading from bottom up: 0x4CB2C

# Hexidecimal

To convert from hexadecimal to decimal, multiply each of the hexadecimal digits by the appropriate power of 16:

0x7AF:

$$\begin{aligned} &7 * 16^2 + 10 * 16 + 15 \\ &= 7 * 256 + 160 + 15 \\ &= 1792 + 160 + 15 = 1967 \end{aligned}$$



# Hexadecimal: It's funky but concise

- Let's take a byte (8 bits):

165

Base-10: Human-readable,  
but cannot easily interpret on/off bits

0b10100101

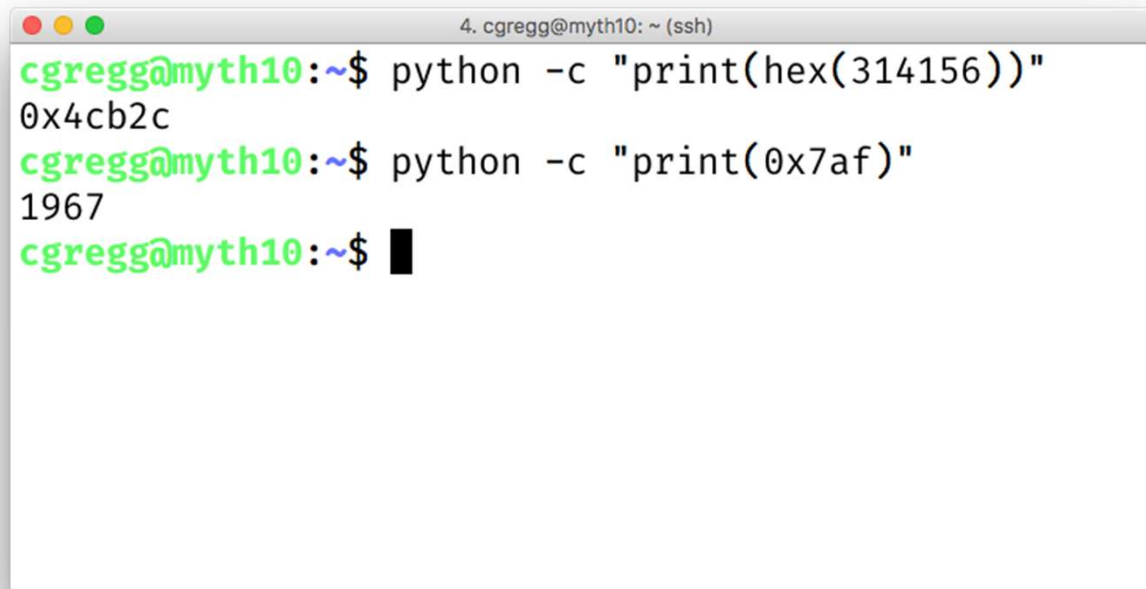
Base-2: Yes, computers use this,  
but not human-readable

0xa5

Base-16: Easy to convert to Base-2,  
More “portable” as a human-readable format  
(fun fact: a half-byte is called a nibble or nybble)

# Let the computer do it!

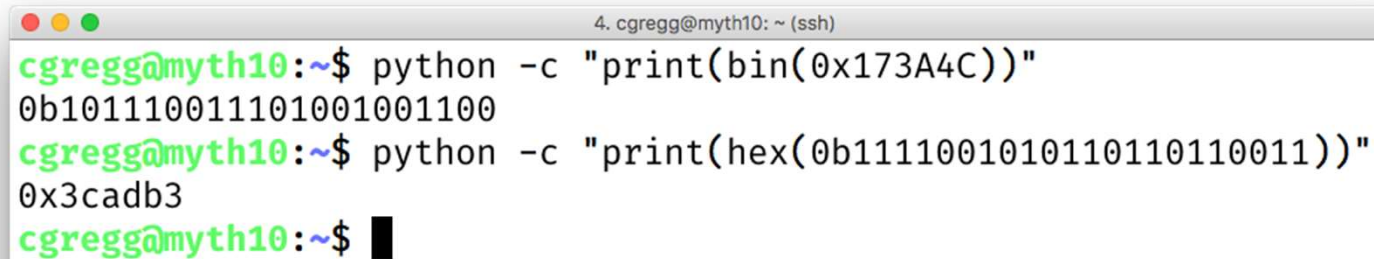
Honestly, hex to decimal and vice versa are easy to let the computer handle. You can either use a search engine (Google does this automatically), or you can use a python one-liner:

A terminal window with a title bar showing '4. cgregg@myth10: ~ (ssh)'. The prompt is 'cgregg@myth10:~\$'. The first command is 'python -c "print(hex(314156))"', which outputs '0x4cb2c'. The second command is 'python -c "print(0x7af)"', which outputs '1967'. The prompt is then followed by a black cursor block.

```
4. cgregg@myth10: ~ (ssh)
cgregg@myth10:~$ python -c "print(hex(314156))"
0x4cb2c
cgregg@myth10:~$ python -c "print(0x7af)"
1967
cgregg@myth10:~$ █
```

# Let the computer do it!

You can also use Python to convert to and from binary:



```
4. cgregg@myth10: ~ (ssh)
cgregg@myth10:~$ python -c "print(bin(0x173A4C))"
0b1011110011101001001100
cgregg@myth10:~$ python -c "print(hex(0b1111001010110110110011))"
0x3cadb3
cgregg@myth10:~$
```

(but you should memorize this as it is easy and you will use it frequently)