CSCI 141

Lecture 5:
More on print and input
Operator Precedence
Binary representation

Announcements

- Academic Honesty and googling for answers:
  - Searching the internet to learn about Python features, syntax, etc. **does not** violate academic honesty.
    - Programmers do this all the time.
    - You learned how to solve a problem!
  - Searching the internet for a solution to a problem I've given you and copy/pasting code **does** violate academic honesty.
    - You didn't learn how to solve the problem.

Goals

- Know how to use keyword arguments such as the `sep` and `end` keyword arguments to the `print` function.
- Know how to save a function's return value to a variable.
- Understand how the + operator behaves with string operands.
- Know how to apply operator precedence rules to determine the order in which pieces of an expression are evaluated.
- Know how to convert a decimal number to binary and vice versa.
- Understand the basic idea behind how strings and floating-point numbers are represented on computers.

What have we covered so far?

- Data is (somehow) stored in memory.

  more on this today: representing numbers in binary!

- Each piece of data has a type.

  so far we've seen: `int, float, str`

- Variables can assign names to pieces of data. **the assignment operator stores a value in a variable, as in:**

  ```
  my_var = "Hello, world!"
  ```

- Operators can do things to the data (these operations are performed by the CPU).

  so far: assignment operator (=)

  arithmetic operators: (`+,-,*,/,**,//,%`)

What have we covered so far?

- A function can take inputs (arguments) and can produce an output (return value)

  so far: `input, print, type, int, float, str`

- Statements are instructions that are executed

  so far: assignment statements, such as `my_var = 64 + 8`

- Expressions are like phrases that can be evaluated to determine what value they represent.

  so far:
  - **functions that return values, like `int(42.8)`**
  - **arithmetic expressions, like `(4 + 2) / 2`**
  - **and combinations of other expressions, like `(2**3) // int(user_input)`**

Today's Quiz

- Please write your name at the top:

  Lastname, Firstname

- 4 minutes

Today’s Quiz

- Please write your name at the top:

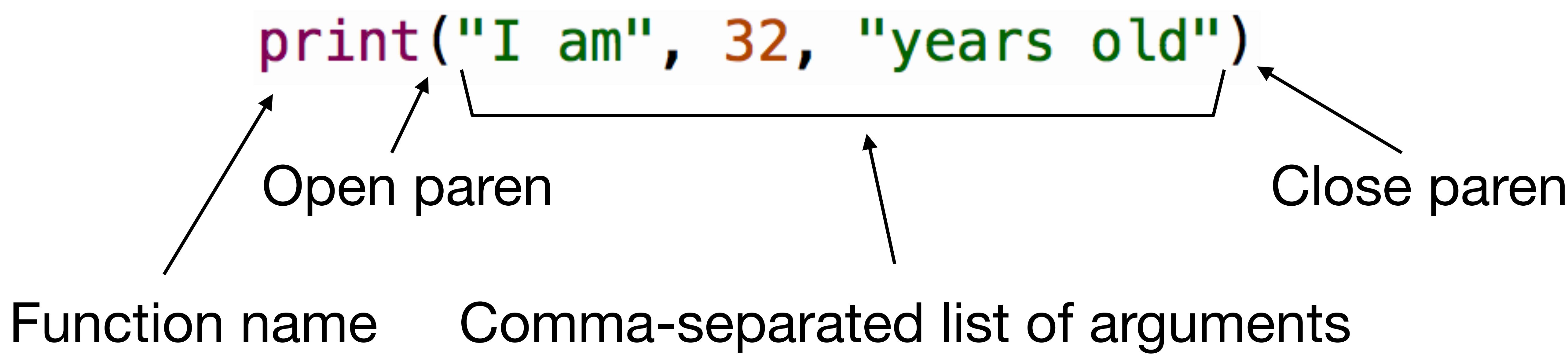
  Lastname, Firstname

- 4 minutes

- Working with a neighbor: do your answers agree? (2 minutes)

Function Calls: Getting Fancier

Syntax for a function call:



Function Calls: Getting Fancier

Keyword arguments provide a way to pass **optional** arguments:

```
print("I am", 31, "years old", sep="")
```

`sep` keyword argument

The `print` function can take two keyword arguments:

- `sep` specifies what goes between the printed arguments (defaults to `sep=" "`)
- `end` specifies what goes after the last printed argument (defaults to `end="\n"`, the character representing a newline)

`input`'s Return Value

The input function waits for the user to enter input on the keyboard:

```
input("Enter some input: ")
```

What if we want to store the input? Use a variable:

```
user_text = input("Enter some input: ")
```

`input`'s return value is whatever text the user entered

Important: `input`'s return value is always returns type `str`

A Note on Operators

- Operators only work if their operands have the correct types.
- Some operators can work on more than one type or combination of types:

Not too surprising:

```
int + int => int
int + float => float
float + int => float
float + float => float
```

Maybe a little surprising:

```
str + str => str
str * int => str
```

Demo

Demo

- print with sep keyword arg
- print with end keyword arg
- save input and convert to an int
- operator behaviors:

```
4 + 5 => 9
4.0 + 5 => 9.0
4.0 + 5.0 => 9.0
“a” + “b” => “ab”
“a” + 1 => error
“a” + “b” => “ab”
“a” * 16 => “aaaaaaaaaaaaaaaa”
```

Order of Operations

We know parenthesized expressions get evaluated from inside to out. Are there any other rules?

What if we took the parentheses out:

```
result = 5 % (3 ** (6 // 4))

result = 5 % 3 ** 6 // 4
```

Order of Operations

We know parenthesized expressions get evaluated from inside to out. Are there any other rules? Yes: operator precedence.

Remember PEMDAS? BIDMAS? BODMAS?

precedence ↑

- Parentheses
- Exponentiation
- Multiplication and Division
- Addition and Subtraction

Order of Operations

We know parenthesized expressions get evaluated from inside to out. Are there any other rules? Yes: operator precedence.

Remember PEMDAS? BIDMAS? BODMAS?

precedence ↑

- Parentheses
- Exponentiation
- Multiplication and Division
- Addition and Subtraction

Example:

```
10 * 6 ** 2 / 5 // 11 - 4
```

Order of Operations

We know parenthesized expressions get evaluated from inside to out. Are there any other rules? Yes: operator precedence.

Remember PEMDAS? BIDMAS? BODMAS?

precedence ↑

Parentheses

Exponentiation

Multiplication and Division (left-to-right)

Addition and Subtraction (left-to-right)

Example:

```
10 * 6 ** 2 / 5 // 11 - 4
```

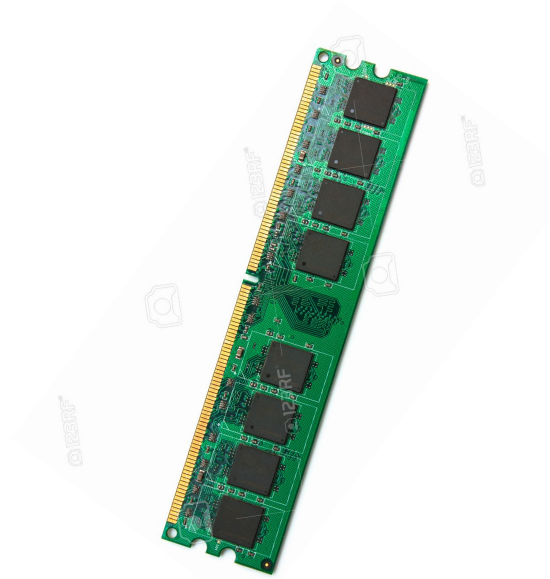
Questions?

Representing Numbers on Computers

- What happens "under the hood" when we execute:

```
result = 5
```

- The value 5 gets stored somewhere in main memory (and we somehow keep track of where it's stored).



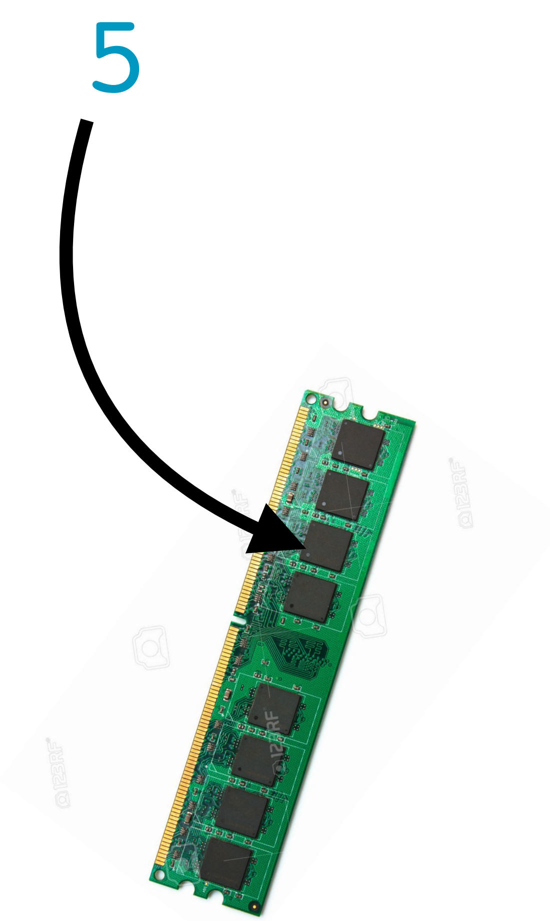
Main Memory

Representing Numbers on Computers

- What happens “under the hood” when we execute:

```
result = 5
```

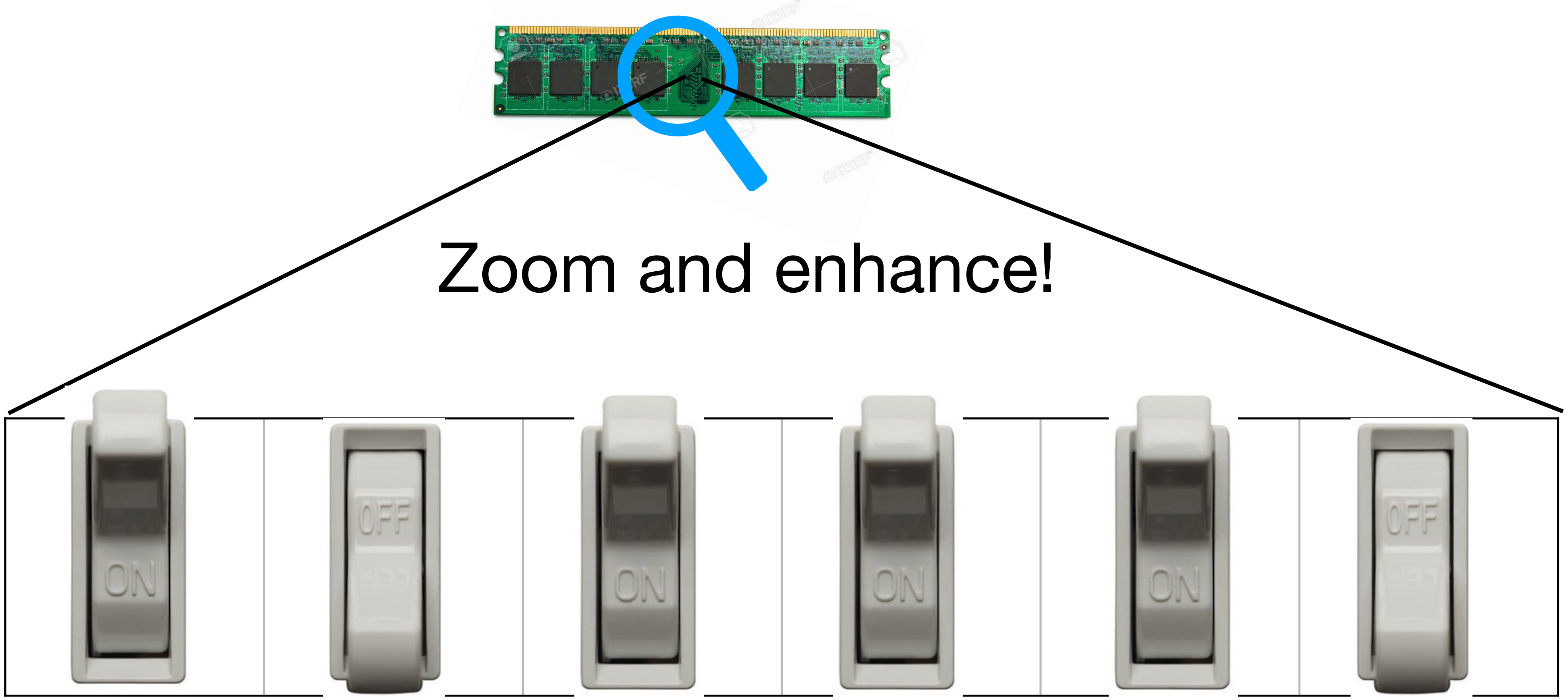
- The value 5 gets stored somewhere in main memory (and we somehow keep track of where it’s stored).



Main Memory

Representing Numbers on Computers

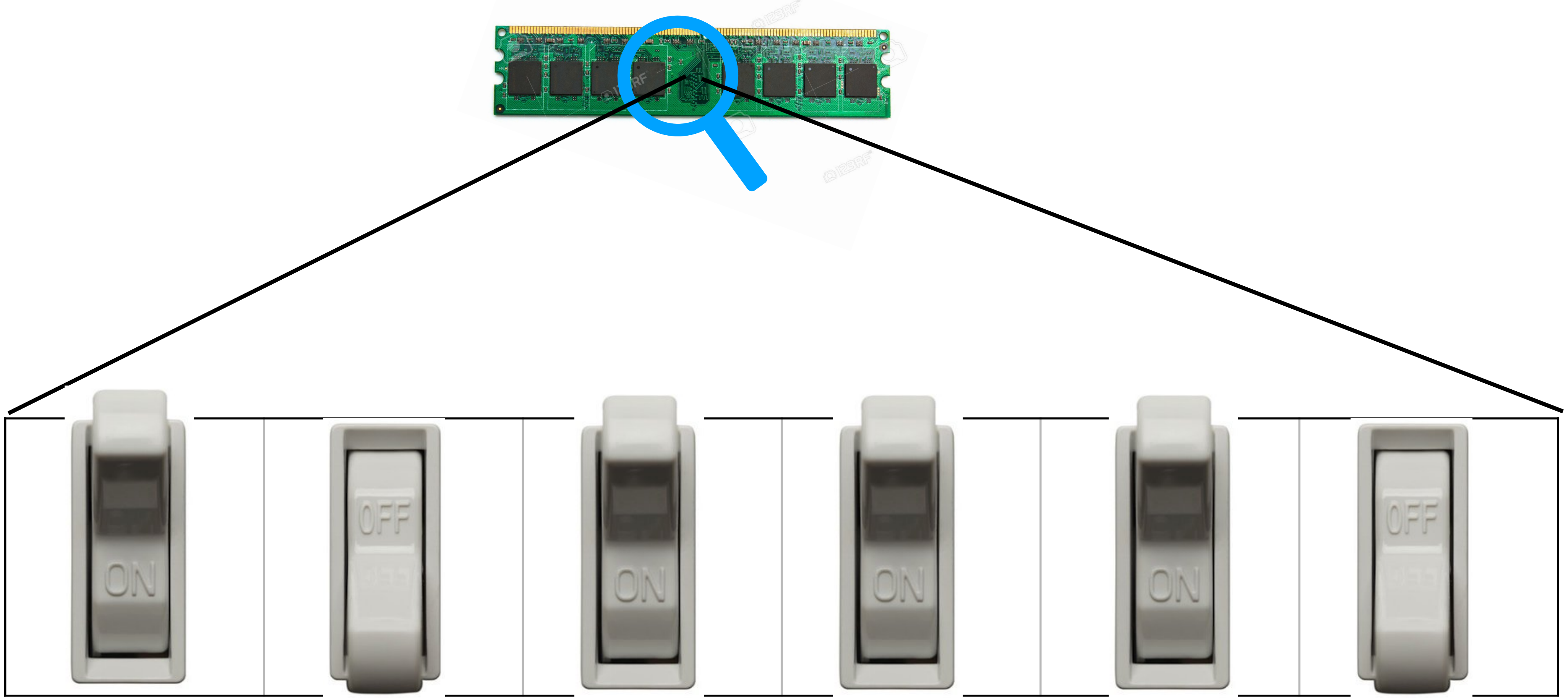
How are numbers stored in memory?



Memory is made of specialized electric circuits that provide cells that can “store” information by being in one of two states: on or off.

Representing Numbers on Computers

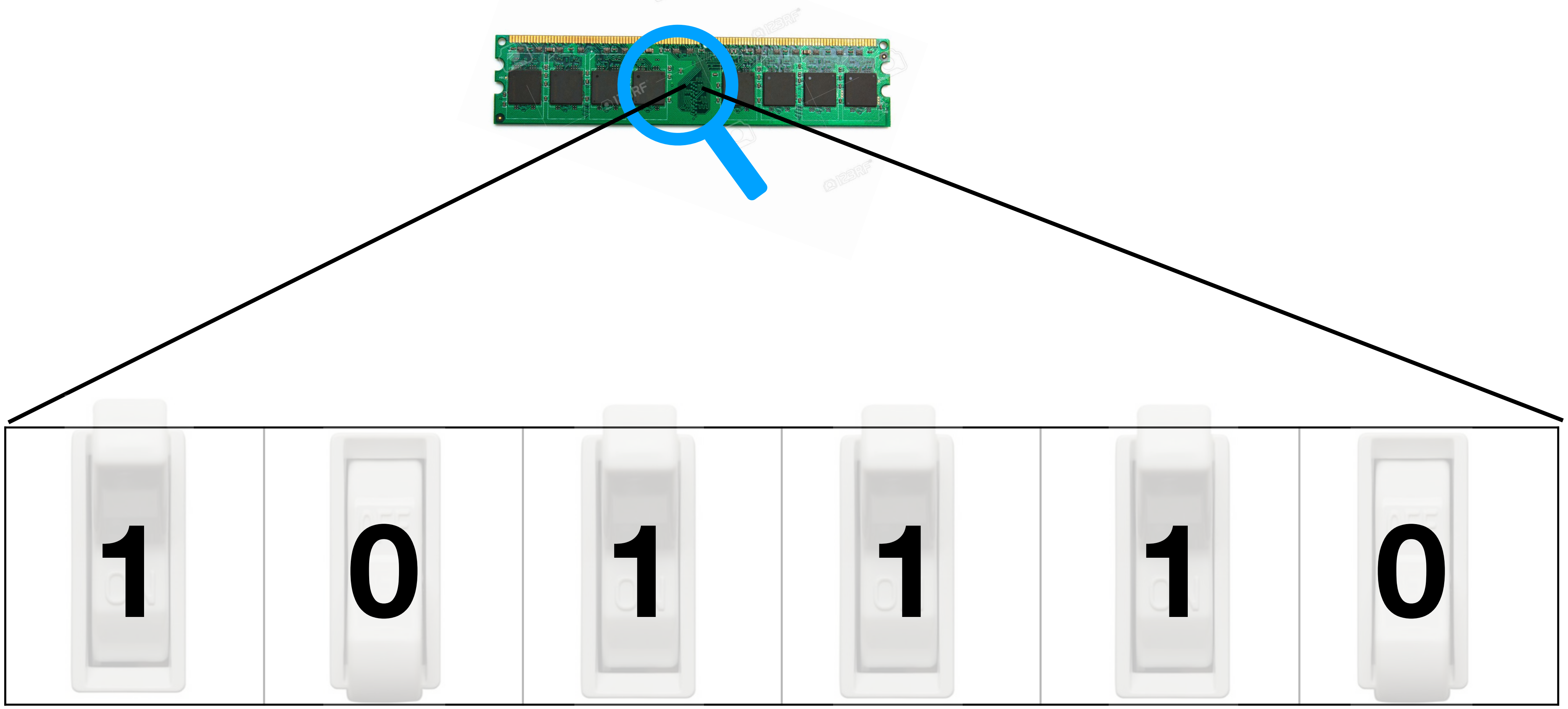
How are numbers stored in memory?



We impose mathematical meaning on these states:
"off" = 0
"on" = 1

Representing Numbers on Computers

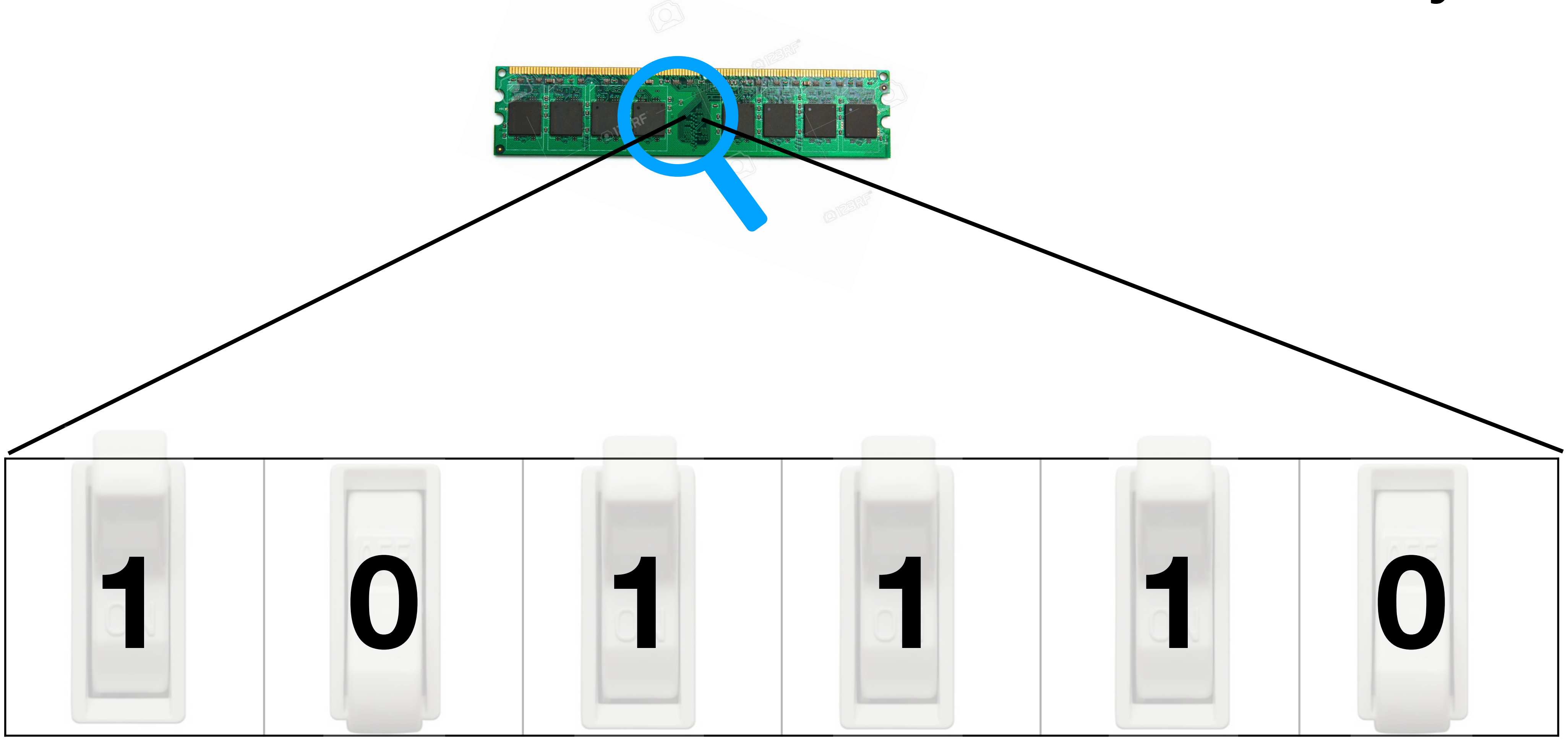
How are numbers stored in memory?



We impose mathematical meaning on these states:
“off” = 0
“on” = 1

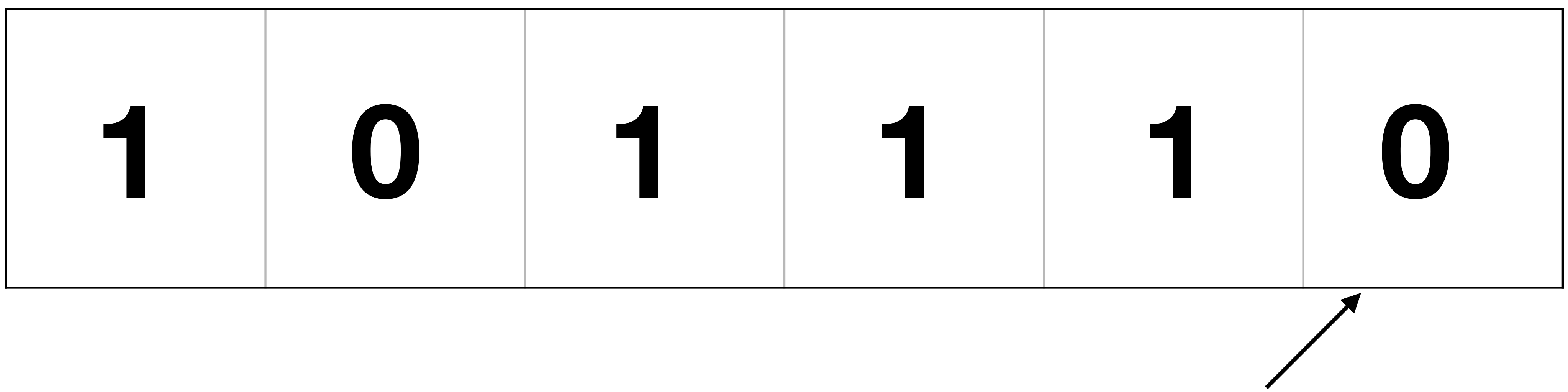
Representing Numbers on Computers

How are numbers stored in memory?



Each 1/0 memory location is called a bit.

Representing Numbers on Computers



Each 0/1 memory location stores one bit.

8 bits is called a byte.

Metric prefixes are used to represent numbers of bytes, e.g. **kilo**, **mega**, **giga**, etc.

In computer science, kilo is not actually 1000, it's 1024.

Representing Numbers on Computers

1	**0**	**1**	**1**	**1**	**0**

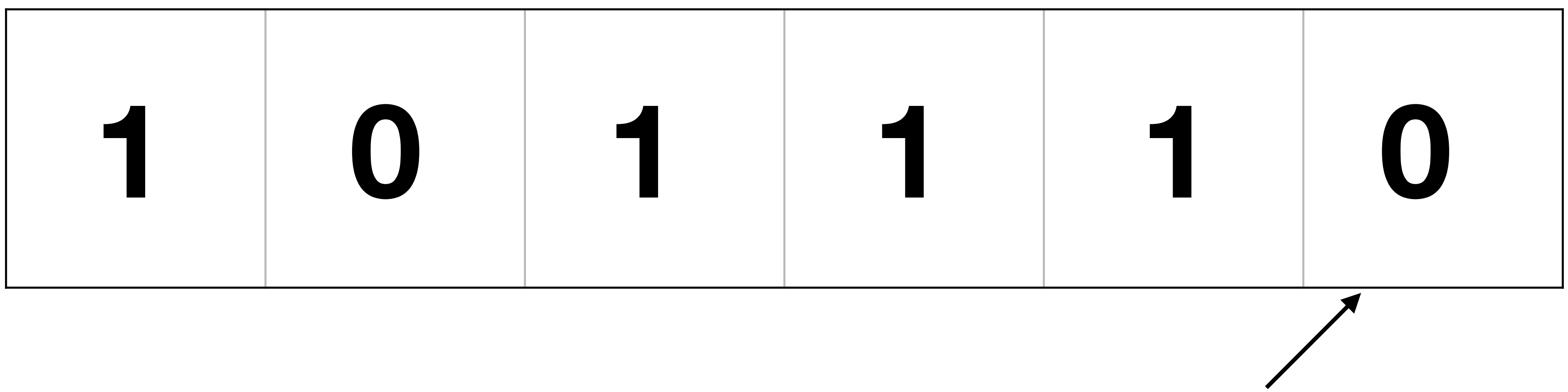
Each 0/1 memory location stores one bit.

8 bits is called a byte.

Metric prefixes are used to represent numbers of bytes, e.g. **kilo**, **mega**, **giga**, etc.

In computer science, the prefixes have slightly different meaning: kilo is not actually 1000, it's 1024.

Representing Numbers on Computers



Each 0/1 memory location stores one bit.

8 bits is called a byte.

Usual SI prefixes:

- kilo = 10^3 = 1000
- mega = 10^6 = 1 million
- giga = 10^9 = 1 billion
- tera = 10^{12} = 1 trillion

Base 2 prefixes:

- kilobyte = 2^{10} = 1,024 bytes
- megabyte = 2^{20} = 1,048,576 bytes
- gigabyte = 2^{30} = 1,073,741,824 bytes
- terabyte = 2^{40} = 1,099,511,627,776 bytes

Binary Representation

If all we can store is 0's and 1's, how do we represent other numbers (e.g., 23?)

- By representing numbers in base 2 (binary) instead of base 10 (decimal).
- Observation:

In decimal:

$$
\begin{aligned}
104 = \; & 1 * 10^2 && \text{(hundreds place)} \\
+ \; & 0 * 10^1 && \text{(tens place)} \\
+ \; & 4 * 10^0 && \text{(ones place)}
\end{aligned}
$$

- The decimal representation of a number is a sum of multiples of the powers of ten.

Binary Representation

If all we can store is 0's and 1's, how do we represent other numbers (e.g., 23?)

- By representing numbers in base 2 (binary) instead of base 10 (decimal).
- Observation:

In decimal:

$$
\begin{aligned}
104 = \; & 1 * 10^2 && \text{(hundreds place)} \\
+ \; & 0 * 10^1 && \text{(tens place)} \\
+ \; & 4 * 10^0 && \text{(ones place)}
\end{aligned}
$$

- Key idea: use 2 here instead of 10.

Binary to Decimal

1	**0**	**1**	**1**	**1**	**1**
2^5	2^4	2^3	2^2	2^1	2^0

- In decimal, each digit represents a multiple of a power of **2**

Binary to Decimal

1	0	1	1	1	1	
2^5	2^4	2^3	2^2	2^1	2^0	
32 +		8 +	4 +	2 +	1	**= 47**

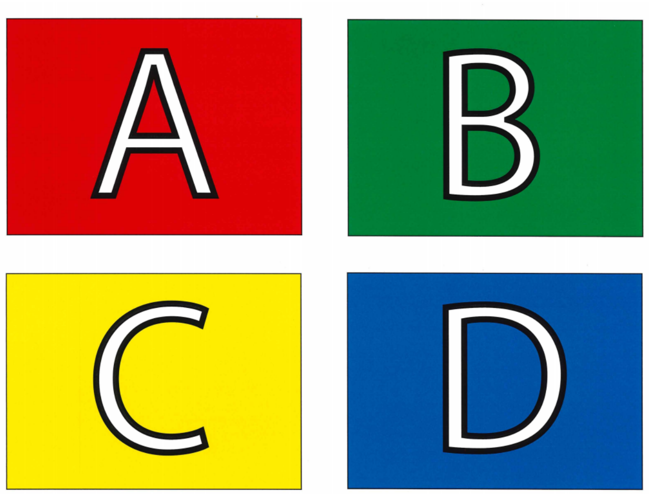
- In decimal, each digit represents a multiple of a power of **2**
- 10111 in binary is 47 in decimal.

Decimal to Binary

Converting decimal to binary goes the other way.

Problem: write 23 as a sum of powers of 2

$$
\begin{aligned}
23 = &\ \boxed{?} * 2^4 \ (16) \\
+ &\ \boxed{?} * 2^3 \ (8) \\
+ &\ \boxed{?} * 2^2 \ (4) \\
+ &\ \boxed{?} * 2^1 \ (2) \\
+ &\ \boxed{?} * 2^0 \ (1)
\end{aligned}
$$



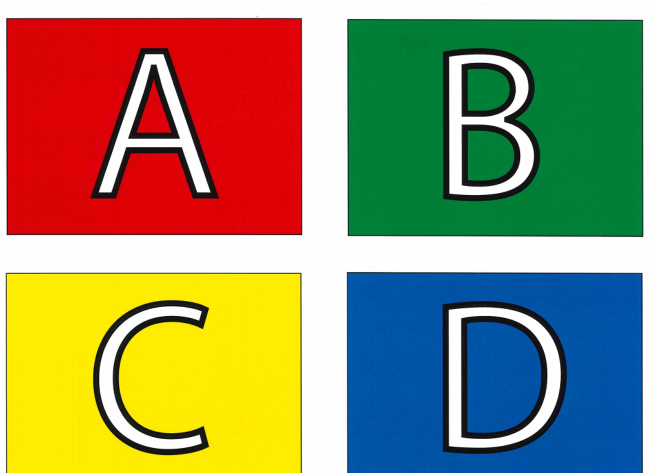
The binary representation of the decimal number 23 is:

A. 10111
B. 11101
C. 01100
D. 11110

Decimal to Binary

Converting decimal to binary goes the other way.
Problem: write 23 as a sum of powers of 2

23 =	? * 2^4	(16)	1	(23-16 = 7 left)
+	? * 2^3	(8)	0	(7-0 = 7 left)
+	? * 2^2	(4)	1	(7-4 = 3 left)
+	? * 2^1	(2)	1	(3-2 = 1 left)
+	? * 2^0	(1)	1	(1-1 = 0 left)



The binary representation of the decimal number 23 is:

A. 10111
B. 11101
C. 01100
D. 11110

That’s how `int` works.

- What about `str` and `float`?

How do you store strings?

Various conventions exist:
ASCII, Unicode

A `str` is a sequence of letters (or characters).

1. Agree by convention on a number that represents each character.
2. Convert that number to binary.
3. Store a sequence of those numbers to form a string.

How do you store strings?

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

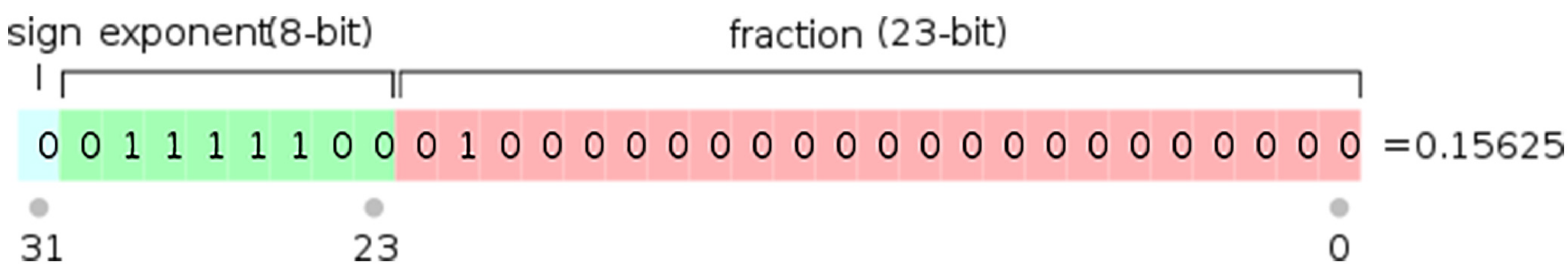
	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
	32	20	[SPACE]	64	40	@	96	60	`
	33	21	!	65	41	A	97	61	a
	34	22	"	66	42	B	98	62	b
	35	23	#	67	43	C	99	63	c
N]	36	24	$	68	44	D	100	64	d
	37	25	%	69	45	E	101	65	e
	38	26	&	70	46	F	102	66	f
	39	27	'	71	47	G	103	67	g
	40	28	(	72	48	H	104	68	h
	41	29	)	73	49	I	105	69	i
	42	2A	*	74	4A	J	106	6A	j
	43	2B	+	75	4B	K	107	6B	k
	44	2C	,	76	4C	L	108	6C	l
	45	2D	-	77	4D	M	109	6D	m
	46	2E	.	78	4E	N	110	6E	n
	47	2F	/	79	4F	O	111	6F	o
	48	30	0	80	50	P	112	70	p
	49	31	1	81	51	Q	113	71	q
	50	32	2	82	52	R	114	72	r
	51	33	3	83	53	S	115	73	s
	52	34	4	84	54	T	116	74	t
DGE]	53	35	5	85	55	U	117	75	u
	54	36	6	86	56	V	118	76	v
K]	55	37	7	87	57	W	119	77	w
	56	38	8	88	58	X	120	78	x
	57	39	9	89	59	Y	121	79	y
	58	3A	:	90	5A	Z	122	7A	z
	59	3B	;	91	5B	[	123	7B	{
	60	3C	<	92	5C	\	124	7C	|
	61	3D	=	93	5D	]	125	7D	}

That's how `str` works.

- What about `float`?
- It's harder to write 4.3752 as a sum of powers of two.

That's how `str` works.

- Floating-point numbers are stored similarly to scientific notation: **1399.94 = 1.39994 * 10^3**
- Need to store the base **and** the exponent. In memory, it looks something like this:



- Base and exponent are represented as base-2 integers, so the precision is finite: not all numbers can be represented!

Exercises

- Convert 1010101 to decimal.
- Convert 1023 to binary.

Next week

Making decisions:

`if` statements and boolean logic.