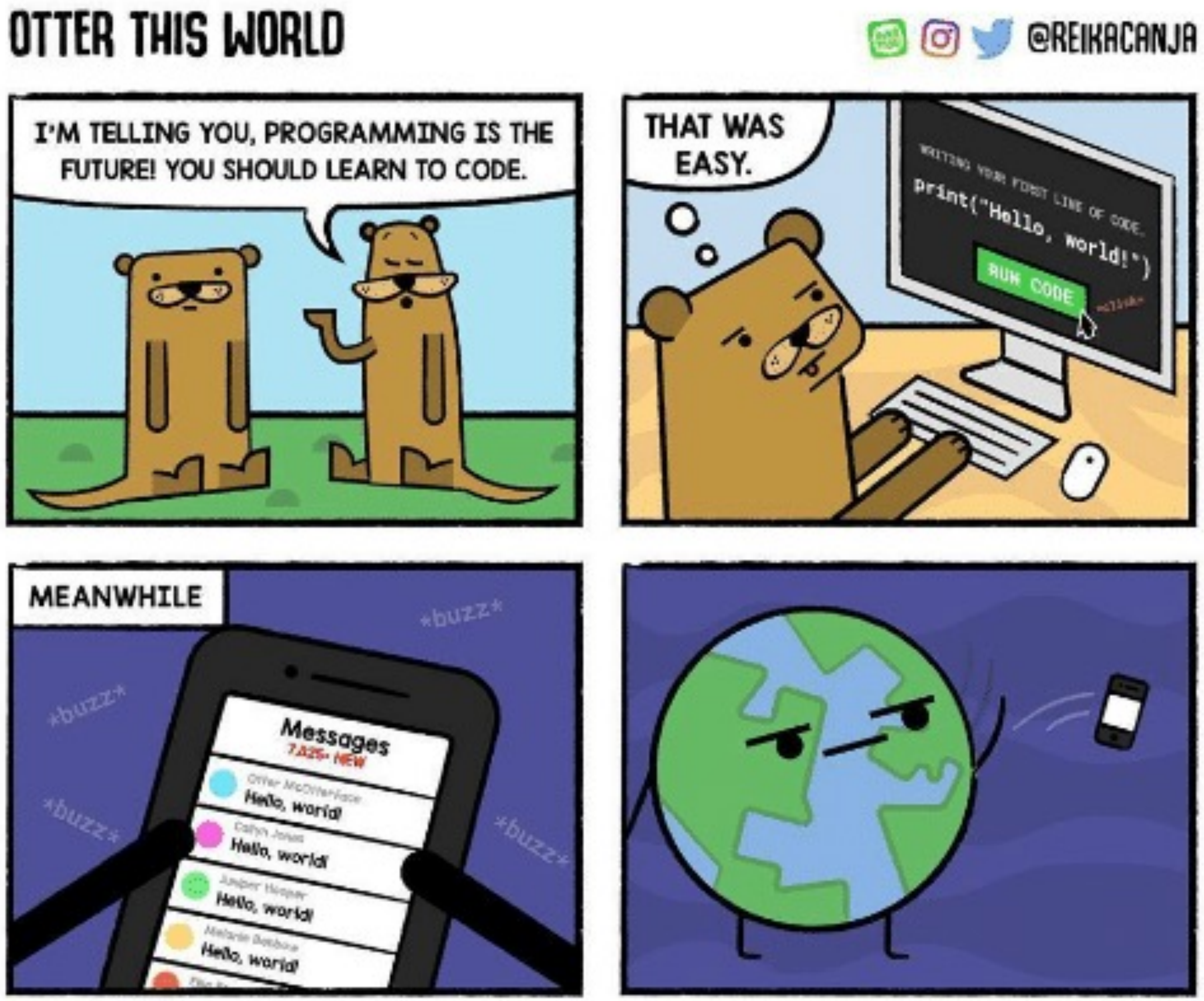
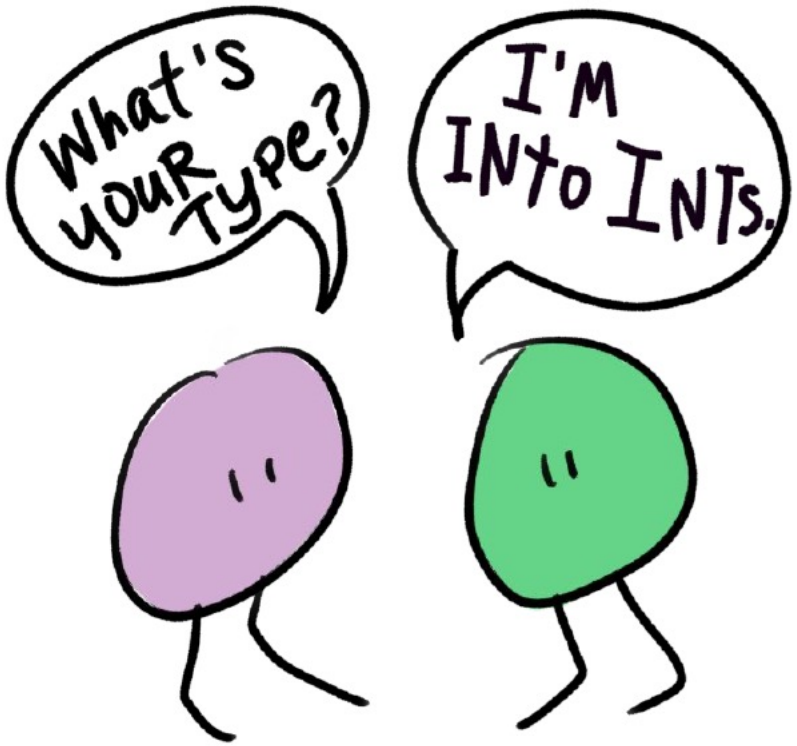


The reason why world never
says hello back!!

EVERYONE SAYS
HELLO WORLD
NO ONE ASKS
HOW IS WORLD?



Variables and Arithmetic

Due this week

- **Recitation 1**
    - Write pseudocode
- **Homework 1**
    - Submit pdf file on Canvas. PDF
- Check the due date! **No late submissions!!**

Today

- Mental Models: how do computers work?
- Variables
- Errors

How do computers work?

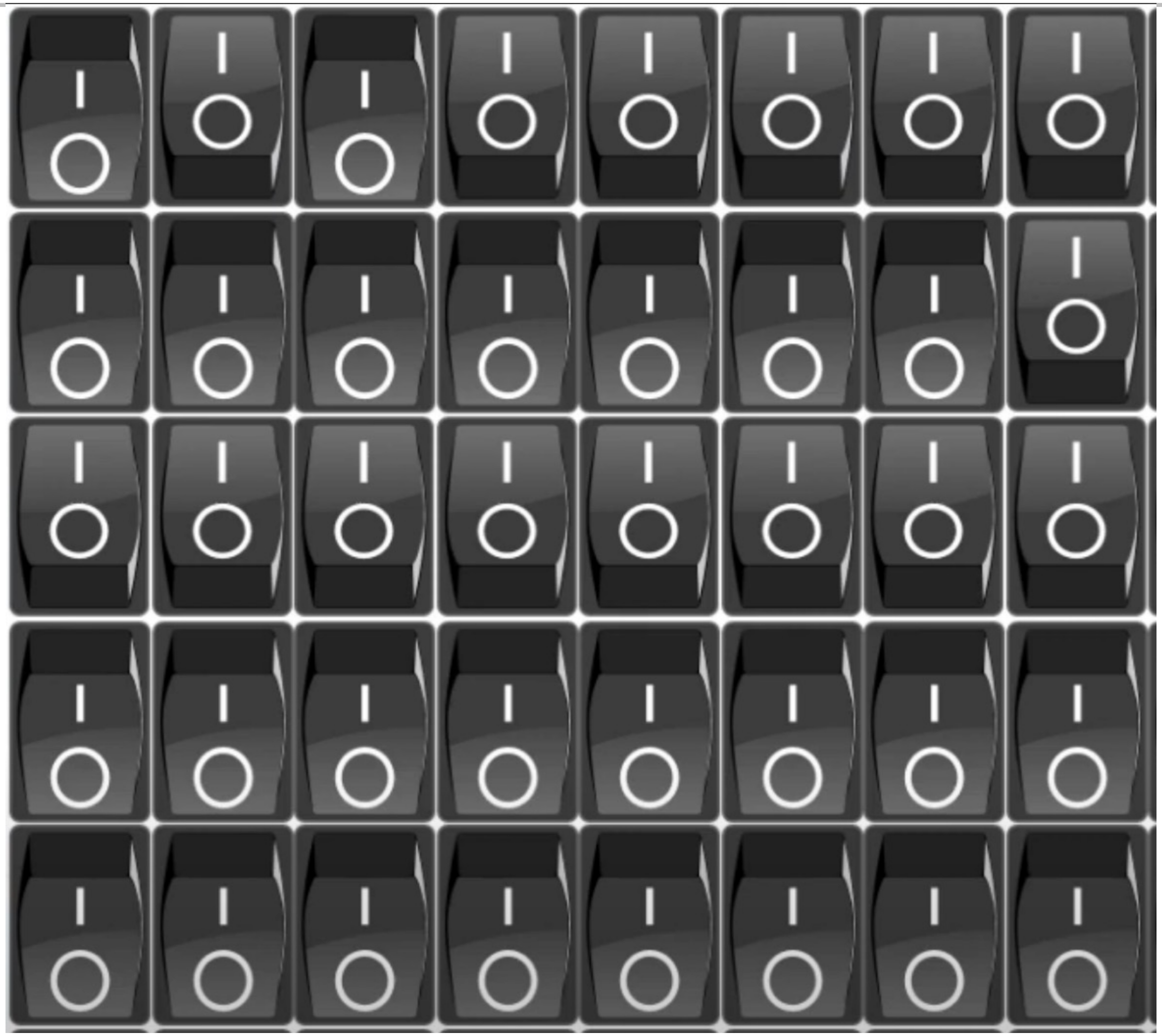
Computers are Walls of Switches

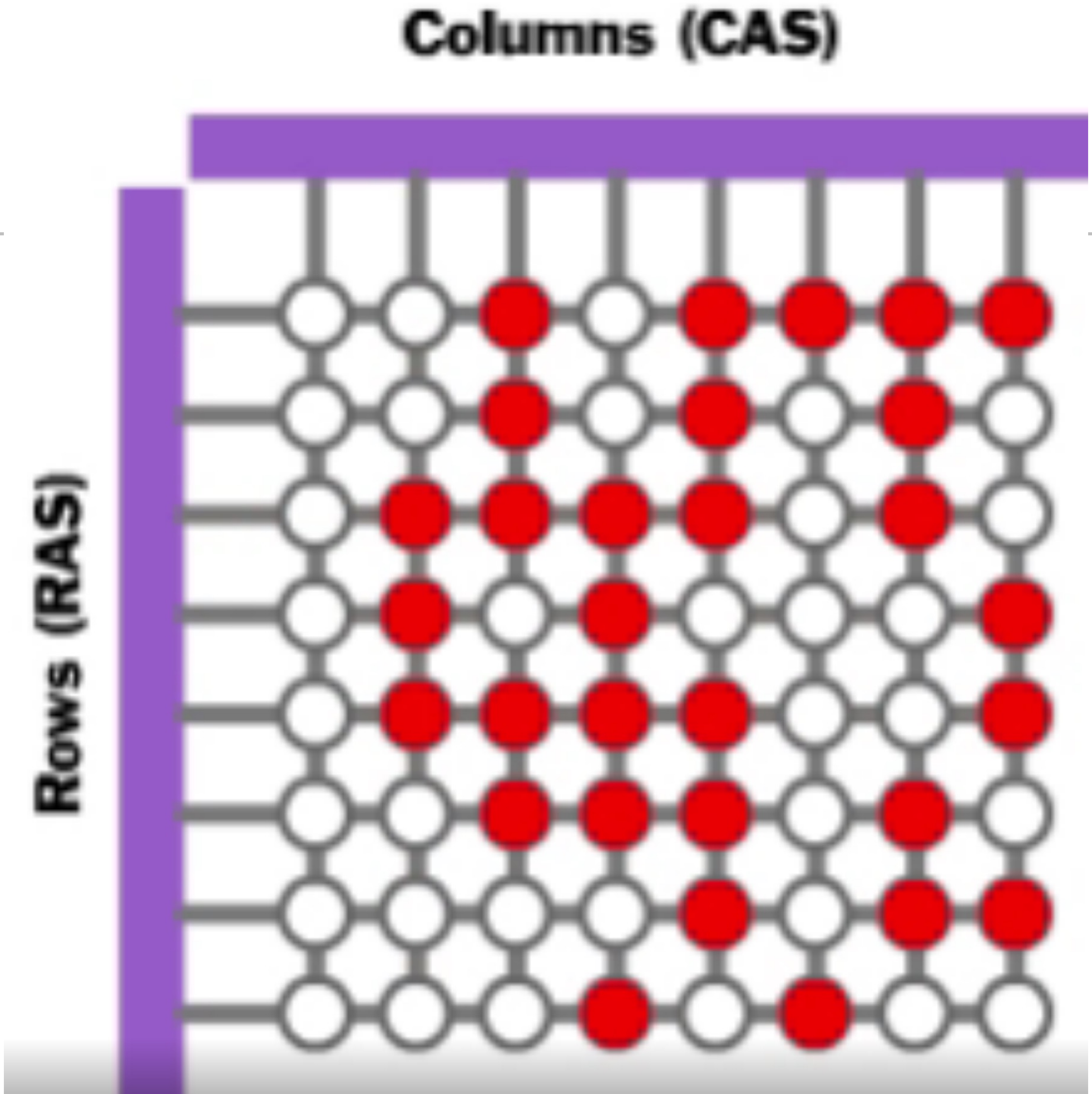
- RAM:

Random

Access

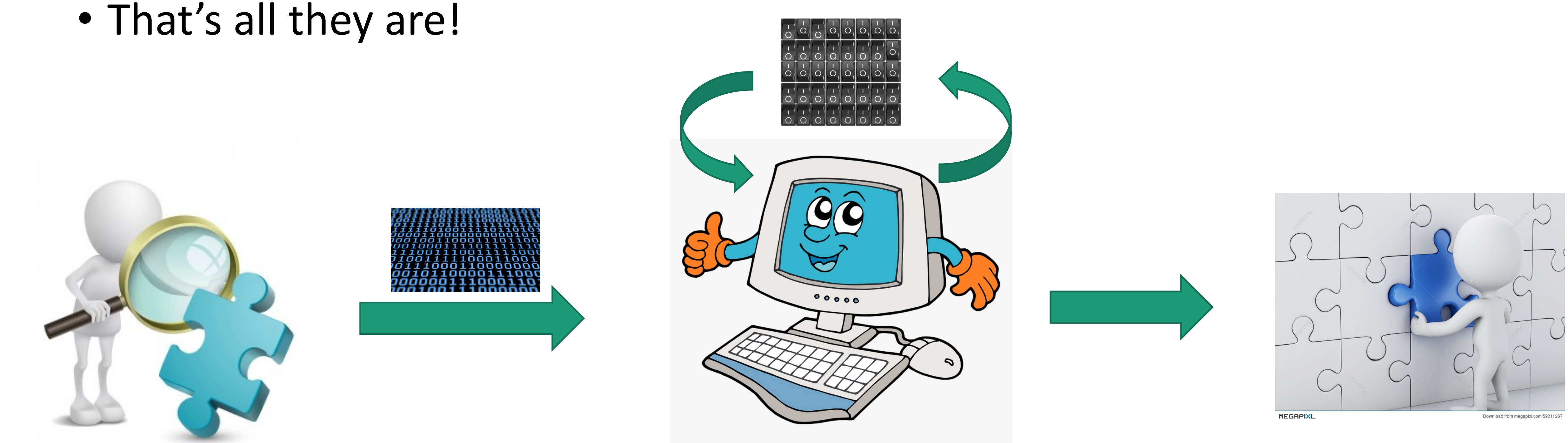
Memory




Columns (CAS)
Rows (RAS)

Computers are Walls of Switches

- That's all they are!



Something of interest in the real world (a scene, an object, a person, etc

Encode it so a computer can understand it (0's and 1's)

Have the happy computer do something to the representation

Decode the bits back to something human-understandable

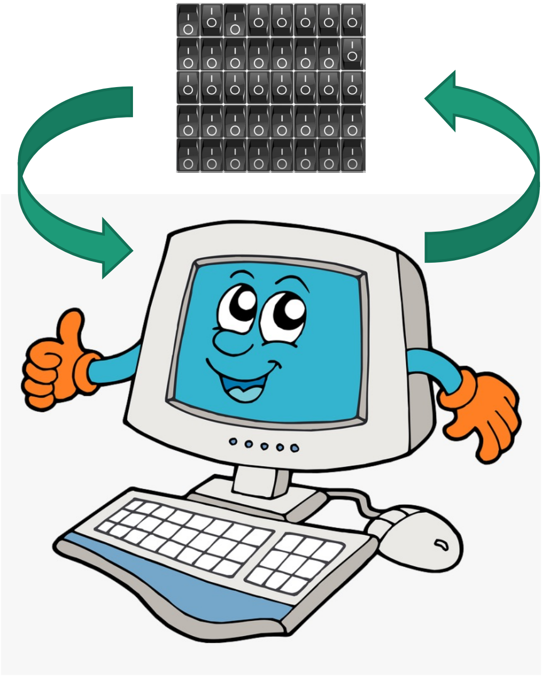
Effect change in the real world

Computers are Walls of Switches

- That’s all they are!

Something of interest in the real world (a scene, an object, a person, etc

Encode it so a computer can understand it (0’s and 1’s)



Have the happy computer do something to the representation (following our programs)

Decode the bits back to something human-understandable

Effect change in the real world

Encoding “Stuff” in Memory

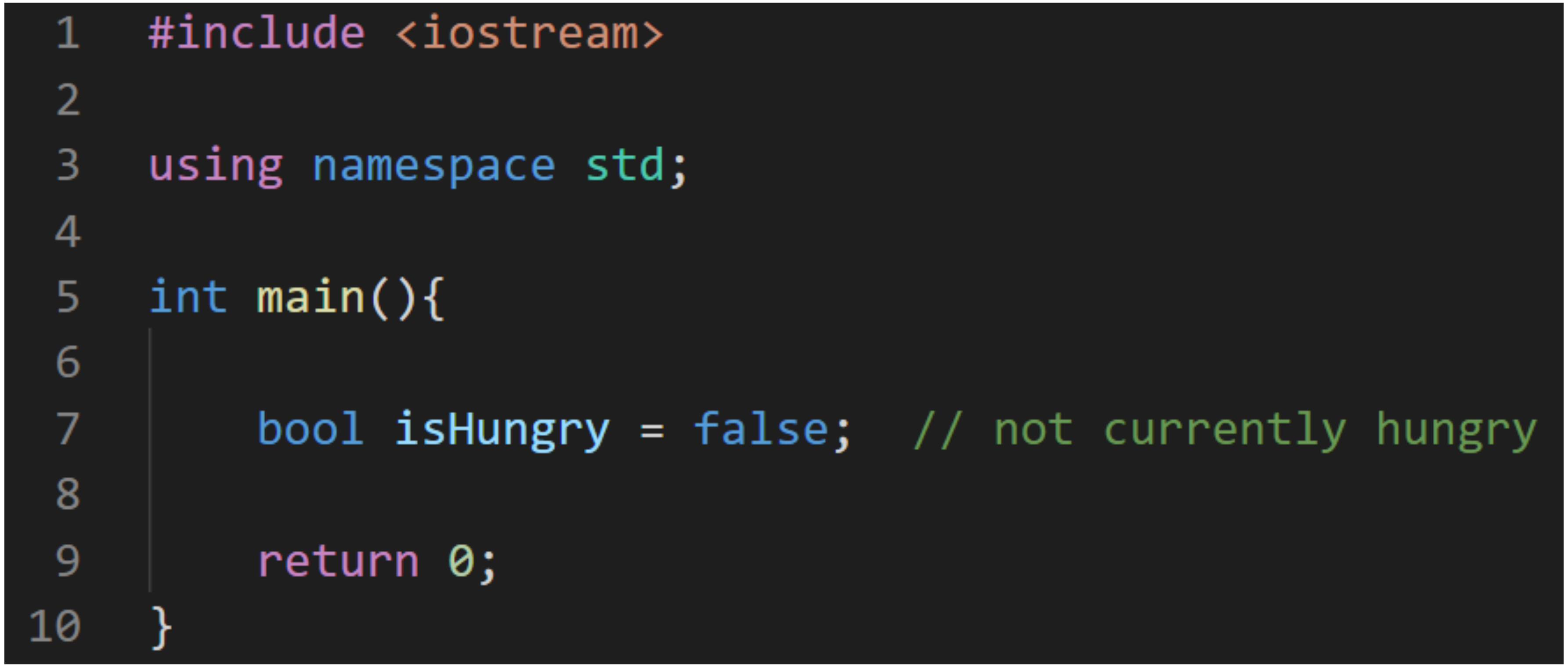
- If a computer is only a bunch of switches (a bunch of 0’s and 1’s)... how can it make sense of anything?

- Let’s start with a small example, and build up

A Single Switch

- One bit of memory. A single 0 or 1
- Could store something useful
  - If there is a fire or not
  - If I am hungry or not
- Represents something as being true or false
- In C++, this is a *boolean* (bool) data type

A Single Switch



```
 1  #include <iostream>
 2
 3  using namespace std;
 4
 5  int main(){
 6
 7      bool isHungry = false;  // not currently hungry
 8
 9      return 0;
10  }
```

- Shhhhh line 7 actually uses 8 bits! (one byte) ... because our memory addresses are per byte and not per bit. Oh well. Memory is cheap now

What about numbers?

- Easy!

- I can just store a zero by using 0 (switch is off)

- And I can store a 1 by using 1 (switch is on)

What about numbers BIGGER than 1?

- Huh

- What to do

What about numbers BIGGER than 1?

- What if I used multiple switches?

- And count up the number of “on” switches?

- THIS COULD WORK
  - ON-ON (1+1) would be 2
  - ON-ON-ON (1+1+1) would be 3
  - ON-ON-ON-ON (1+1+1+1) would be 4

What’s wrong with this?

What’s wrong with just adding up the 1’s?

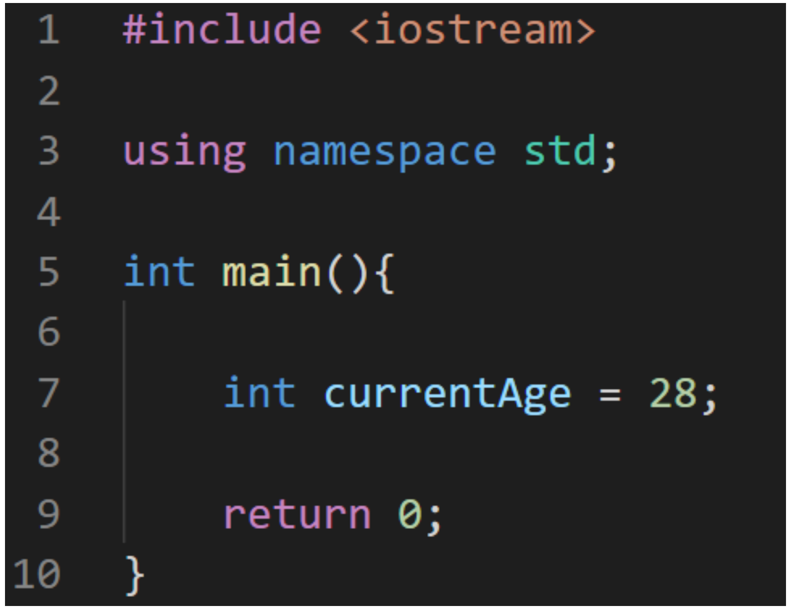
- It throws away memory.
  - 00111 would be the same as 11001
- We could come up with a system that makes *every change meaningful*
- *It’s called binary!*

Number (decimal)	Binary	Our weird system
0	0	0
1	1	1
2	10	11
3	11	111
4	100	1111
5	101	11111
…	…	…
255	11111111	111111111111111111......
256	100000000	111111111111111111...1

Cool! We can now store any* number in memory

- This is huge.

- *In C++, we generally use 8 byte (64 bits) ints (integers)*



```
#include <iostream>

using namespace std;

int main(){

    int currentAge = 28;

    return 0;
}
```

*decimals, anyone?

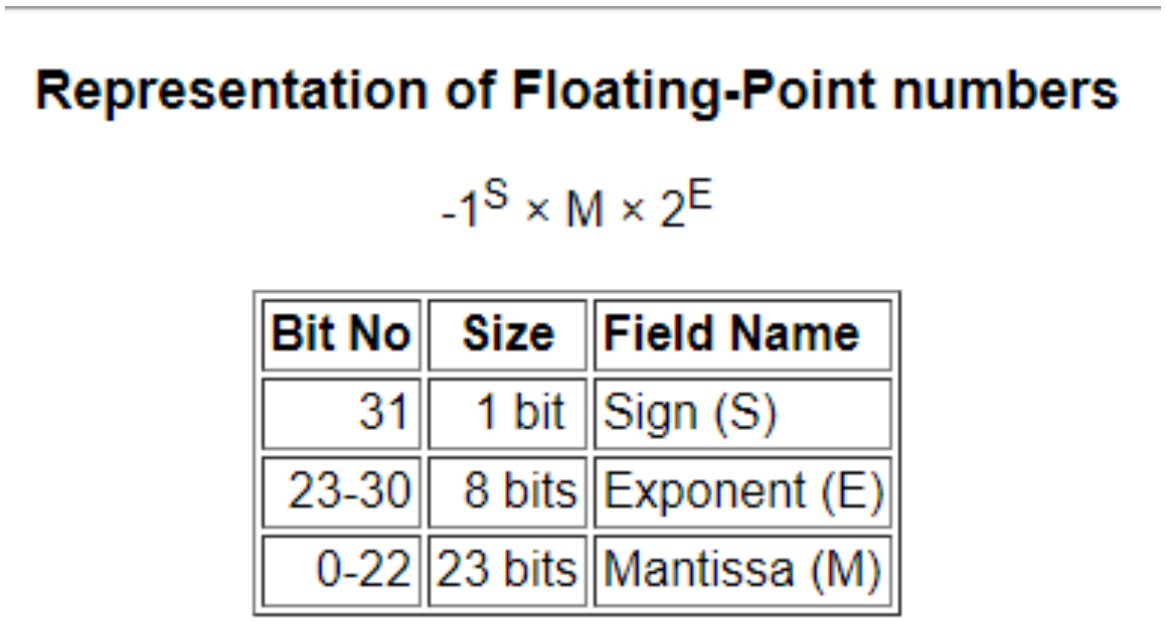
Storing decimals (floating point) in memory

- How can I store 0.5 in memory?

- This is a challenge...

- *Fortunately we can store two integers and turn them into a decimal*

- This is in the weeds... but know that floating point representations are NOT EXACT



Representation of Floating-Point numbers

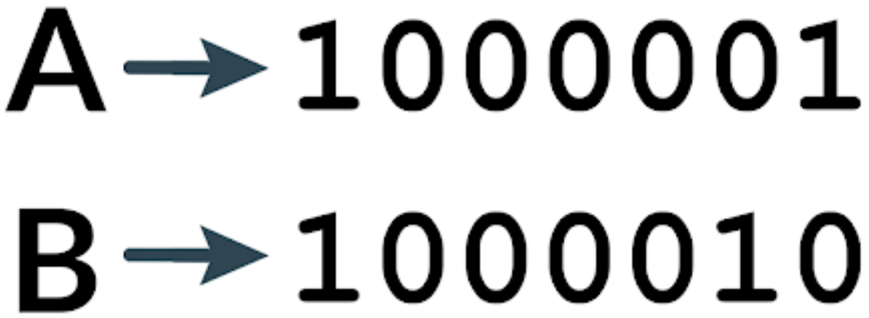
$$-1^S \times M \times 2^E$$

Bit No	Size	Field Name
31	1 bit	Sign (S)
23-30	8 bits	Exponent (E)
0-22	23 bits	Mantissa (M)

What about… letters?

- *Well, we already know how to store whole numbers… let’s just do that*

65 in binary



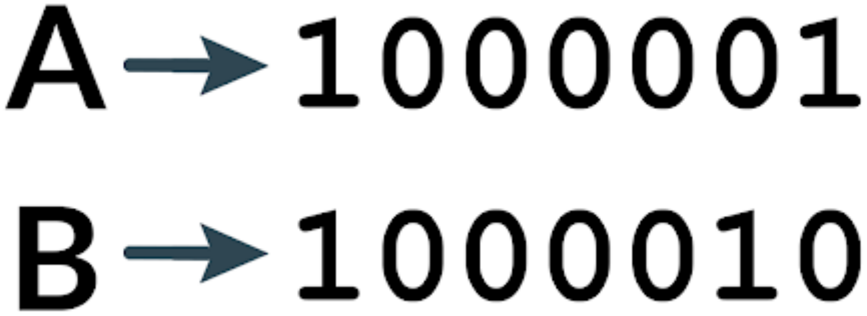
66 in binary

Storing letters in memory

- *Well, we already know how to store numbers... let's just do that*

- 65 ????

65 in binary

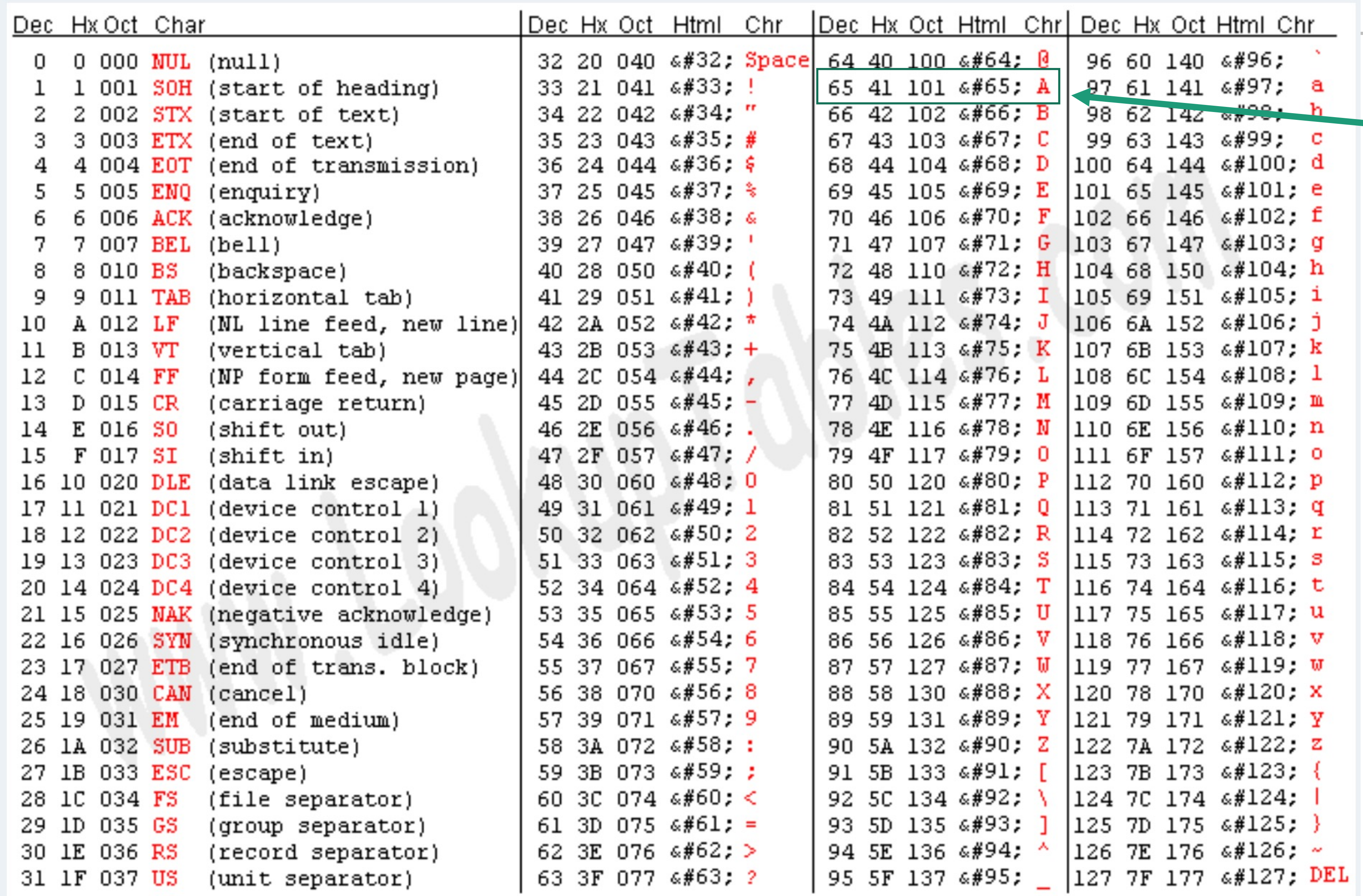


A → 1000001

B → 1000010

66 in binary

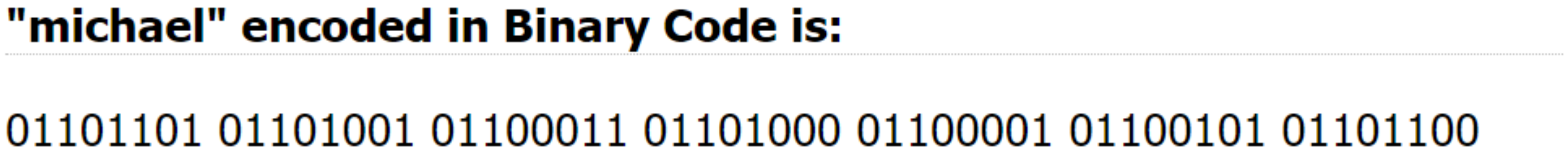
Storing letters in memory – the ASCII table



Dec	Hx	Oct	Char		Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL	(null)	32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	SOH	(start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	STX	(start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	ETX	(end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	EOT	(end of transmission)	36	24	044	&#36;	$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	ENQ	(enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	ACK	(acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
7	7	007	BEL	(bell)	39	27	047	&#39;	'	71	47	107	&#71;	G	103	67	147	&#103;	g
8	8	010	BS	(backspace)	40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
9	9	011	TAB	(horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
10	A	012	LF	(NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
11	B	013	VT	(vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
12	C	014	FF	(NP form feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
13	D	015	CR	(carriage return)	45	2D	055	&#45;	-	77	4D	115	&#77;	M	109	6D	155	&#109;	m
14	E	016	SO	(shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
15	F	017	SI	(shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
16	10	020	DLE	(data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p
17	11	021	DC1	(device control 1)	49	31	061	&#49;	1	81	51	121	&#81;	Q	113	71	161	&#113;	q
18	12	022	DC2	(device control 2)	50	32	062	&#50;	2	82	52	122	&#82;	R	114	72	162	&#114;	r
19	13	023	DC3	(device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	S	115	73	163	&#115;	s
20	14	024	DC4	(device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	T	116	74	164	&#116;	t
21	15	025	NAK	(negative acknowledge)	53	35	065	&#53;	5	85	55	125	&#85;	U	117	75	165	&#117;	u
22	16	026	SYN	(synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	V	118	76	166	&#118;	v
23	17	027	ETB	(end of trans. block)	55	37	067	&#55;	7	87	57	127	&#87;	W	119	77	167	&#119;	w
24	18	030	CAN	(cancel)	56	38	070	&#56;	8	88	58	130	&#88;	X	120	78	170	&#120;	x
25	19	031	EM	(end of medium)	57	39	071	&#57;	9	89	59	131	&#89;	Y	121	79	171	&#121;	y
26	1A	032	SUB	(substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	Z	122	7A	172	&#122;	z
27	1B	033	ESC	(escape)	59	3B	073	&#59;	;	91	5B	133	&#91;	[	123	7B	173	&#123;	{
28	1C	034	FS	(file separator)	60	3C	074	&#60;	<	92	5C	134	&#92;	\	124	7C	174	&#124;	|
29	1D	035	GS	(group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	]	125	7D	175	&#125;	}
30	1E	036	RS	(record separator)	62	3E	076	&#62;	>	94	5E	136	&#94;	^	126	7E	176	&#126;	~
31	1F	037	US	(unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	DEL

American
Standard
Code for
Information
Interchange

Storing words and sentences in memory

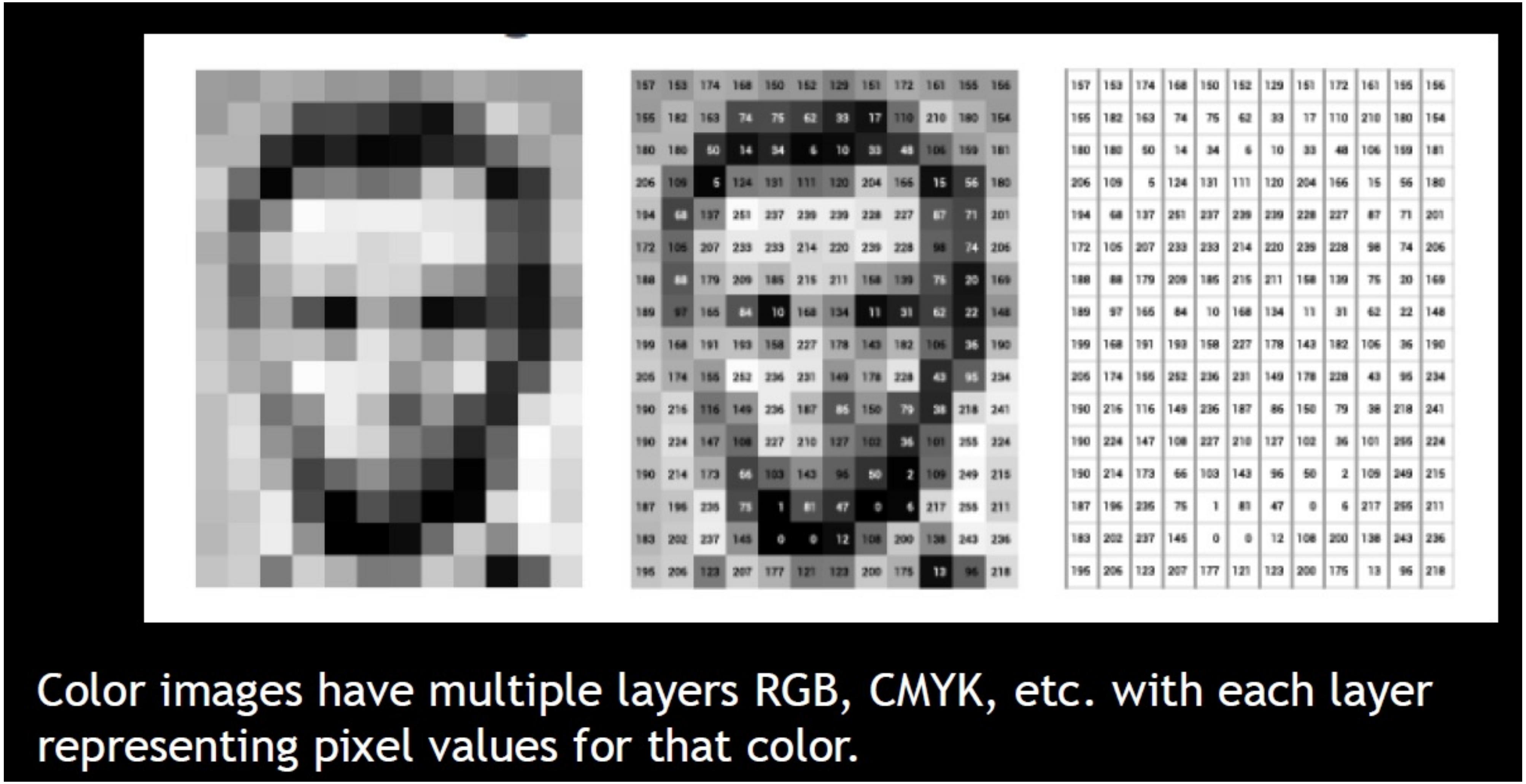


"michael" encoded in Binary Code is:

01101101 01101001 01100011 01101000 01100001 01100101 01101100

m (109) i (105) c (99) h (104) a (97) e (101) l(108)

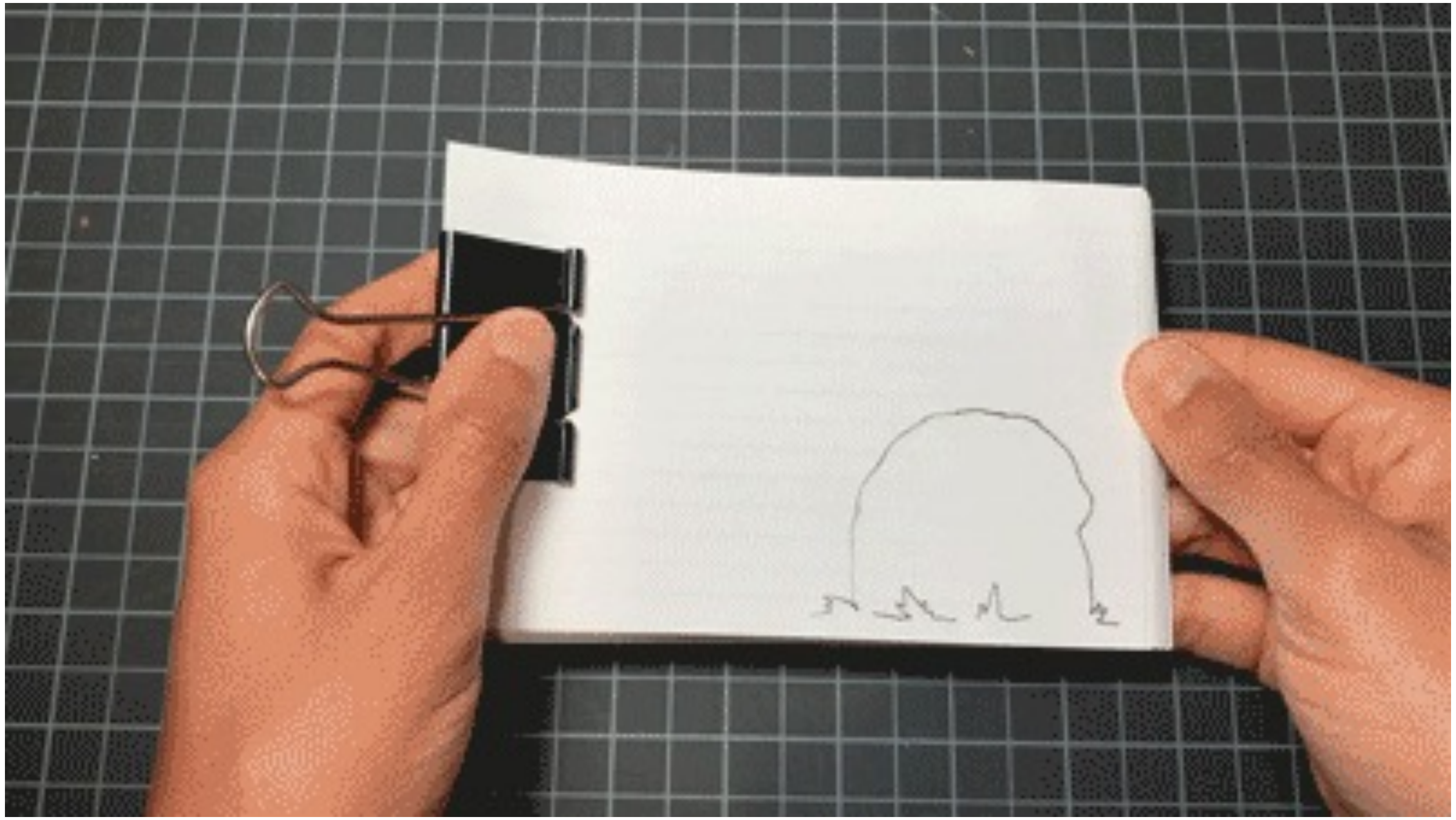
What about... images?



157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	95	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

Color images have multiple layers RGB, CMYK, etc. with each layer representing pixel values for that color.

What about… movies?



What about... people?

- We can pick out things we care about (name, age, etc) and encode those with methods we know how

- We can take photos of people...

- We can 3D scan people and store their physical geometry?

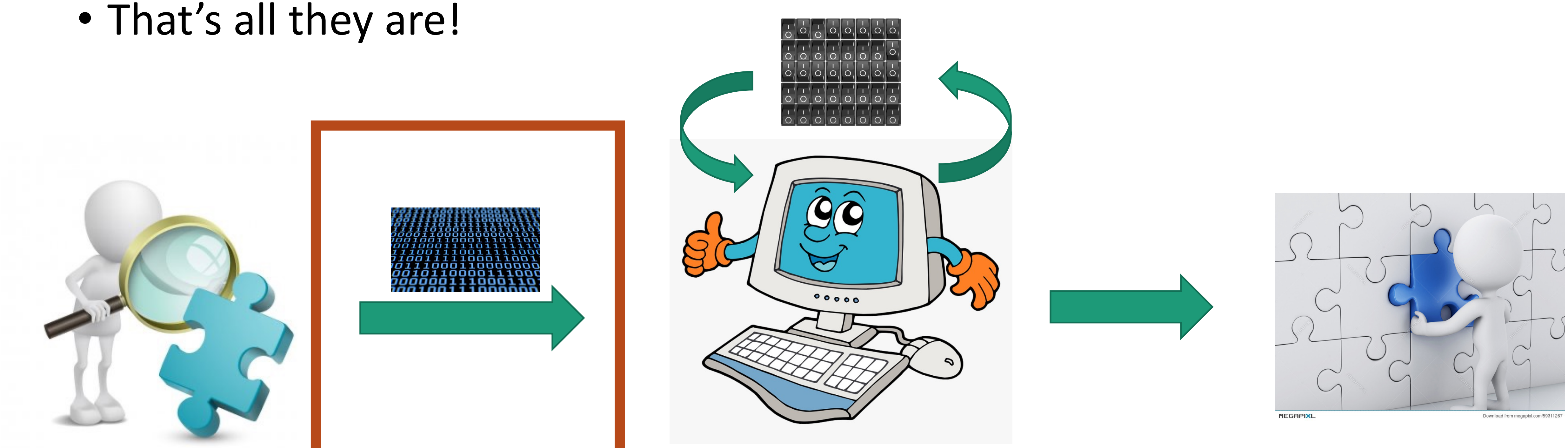
- Can we represent an individual’s mind in a computer?

What about... people?

- What is a “person” to Facebook?
  - Your likes, activity, friends, etc.
- Spotify?
- TikTok?
- The field of AI / Machine learning is trying to do more... store intelligence in a computer
- One of you might figure this out

Computers are Walls of Switches

- That’s all they are!



Something of interest in the real world (a scene, an object, a person, etc

Encode it so a computer can understand it (0’s and 1’s)

Have the happy computer do something to the representation (following our programs)

Decode the bits back to something human-understandable

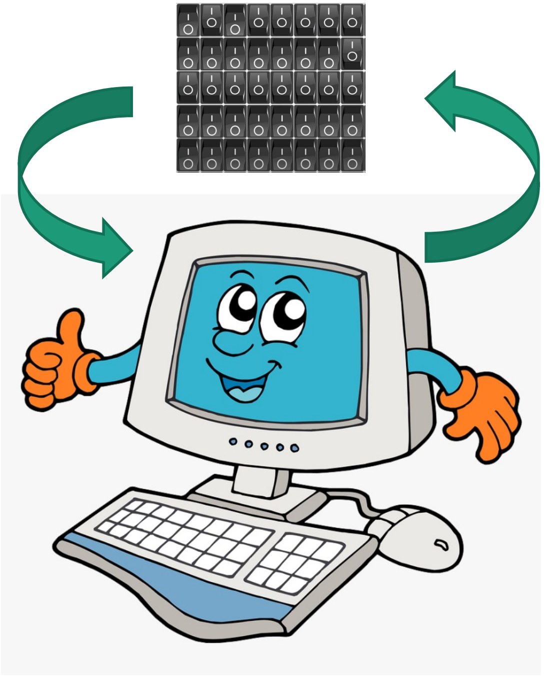
Effect change in the real world

Computers are Walls of Switches

- That’s all they are!

Something of interest in the real world (a scene, an object, a person, etc

Encode it so a computer can understand it (0’s and 1’s)



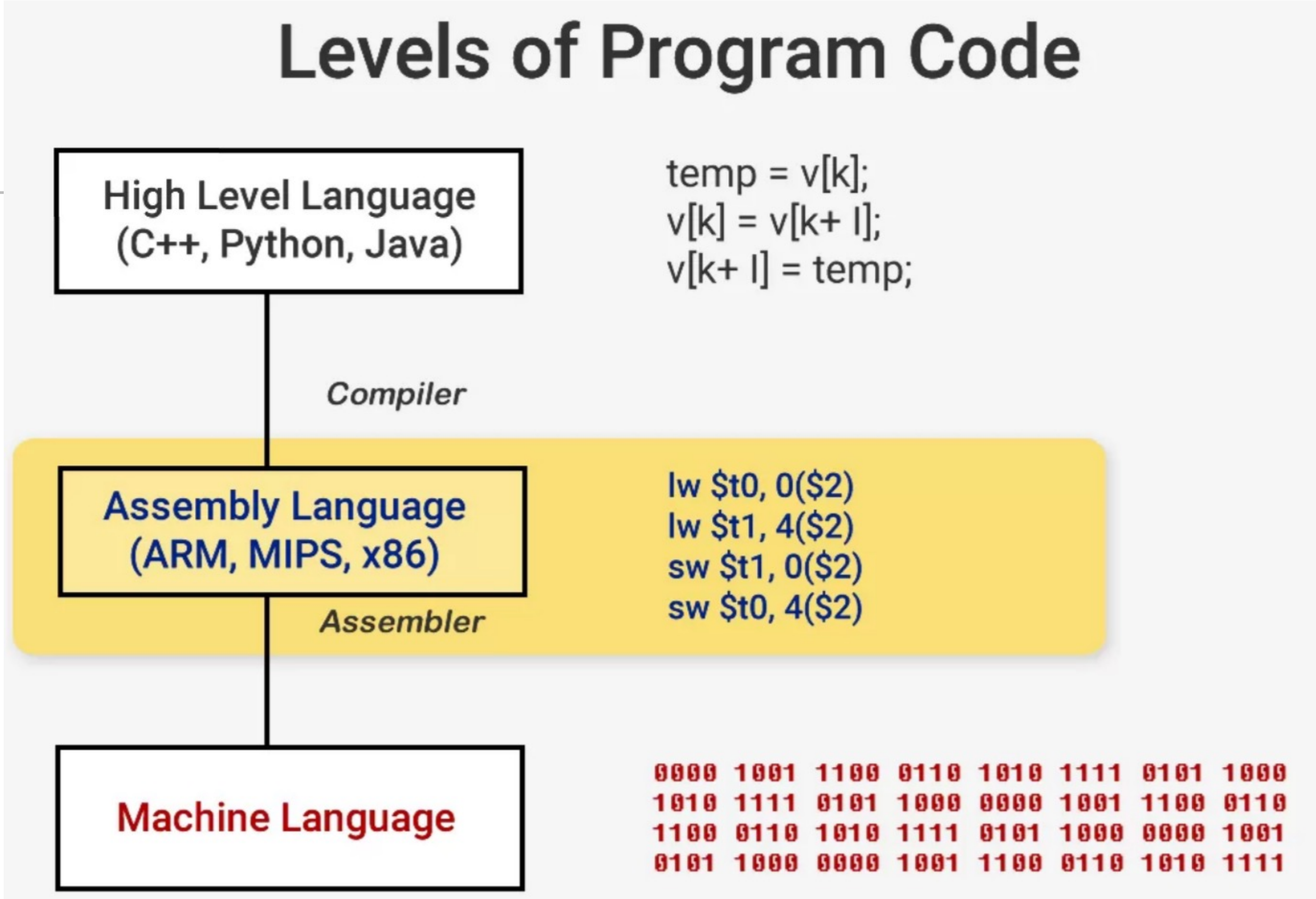
Have the happy computer do something to the representation (following our programs)

Decode the bits back to something human-understandable

Effect change in the real world

How do we give instructions to the computer?

- Programming of course, but computers don’t know C++... (humans do).
- Humans have built tools to translate C++ into computer speak (machine language)... compiler/assembler



Levels of Program Code

High Level Language
(C++, Python, Java)

```
temp = v[k];
v[k] = v[k+ l];
v[k+ l] = temp;
```

Compiler

Assembly Language
(ARM, MIPS, x86)

```
lw $t0, 0($2)
lw $t1, 4($2)
sw $t1, 0($2)
sw $t0, 4($2)
```

Assembler

Machine Language

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

How do we give instructions to the computer?

- We are writing a book for the computer (processor) to read.
- The book contains instructions on which switches should be looked at, which switches should be flipped, and how to organize the switches
- Let’s first look at organizing the wall of switches with variables

Variables

Variables

A **variable:**

- is used to **store** information (the **value**/**contents** of the variable)
    - can contain one piece of information at a time.
- has an **identifier** (the name of the variable)

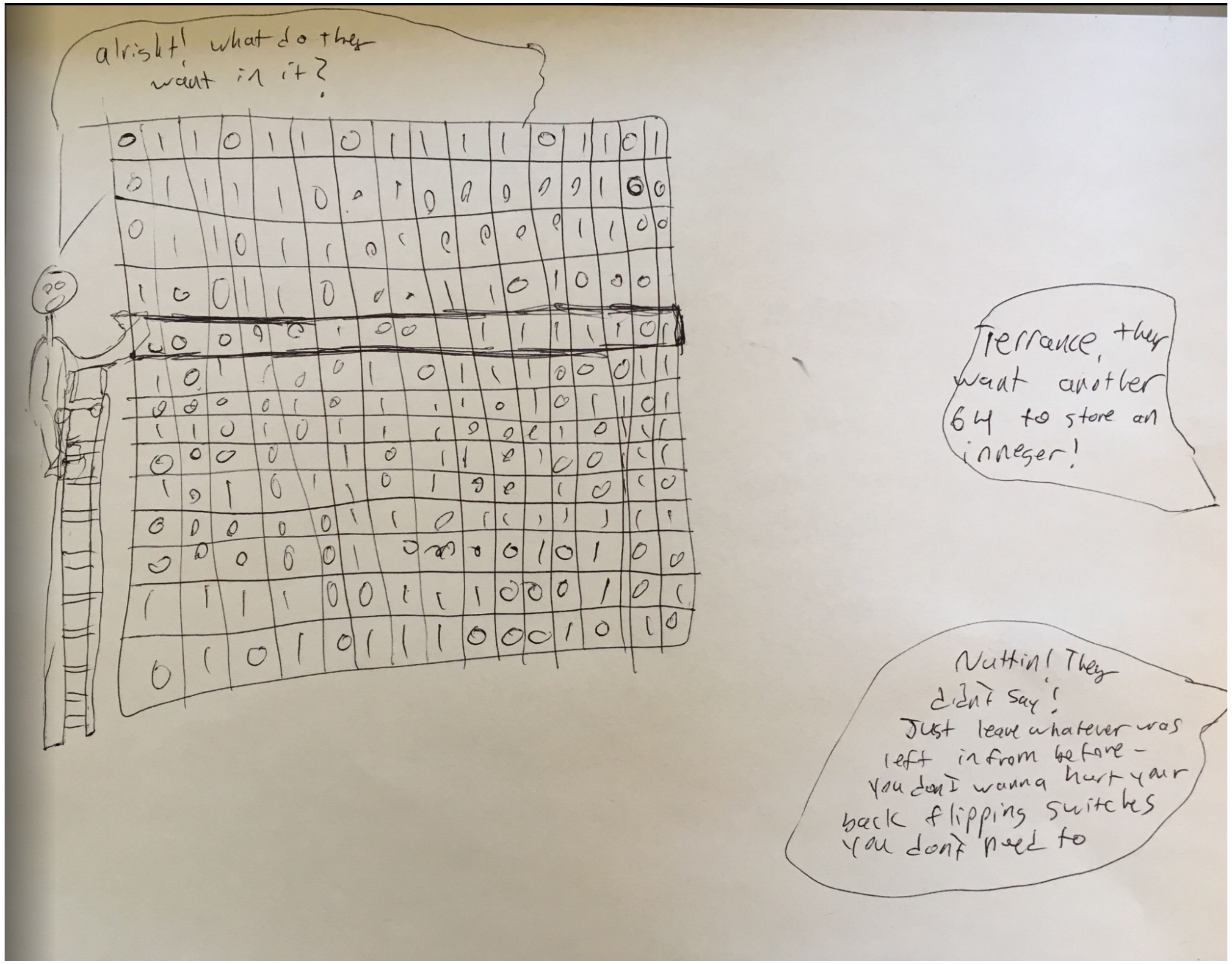
- The programmer picks a good name
    - A good name describes the contents of the variable or what the variable will be used for
    - has a **type** (more about this very soon)

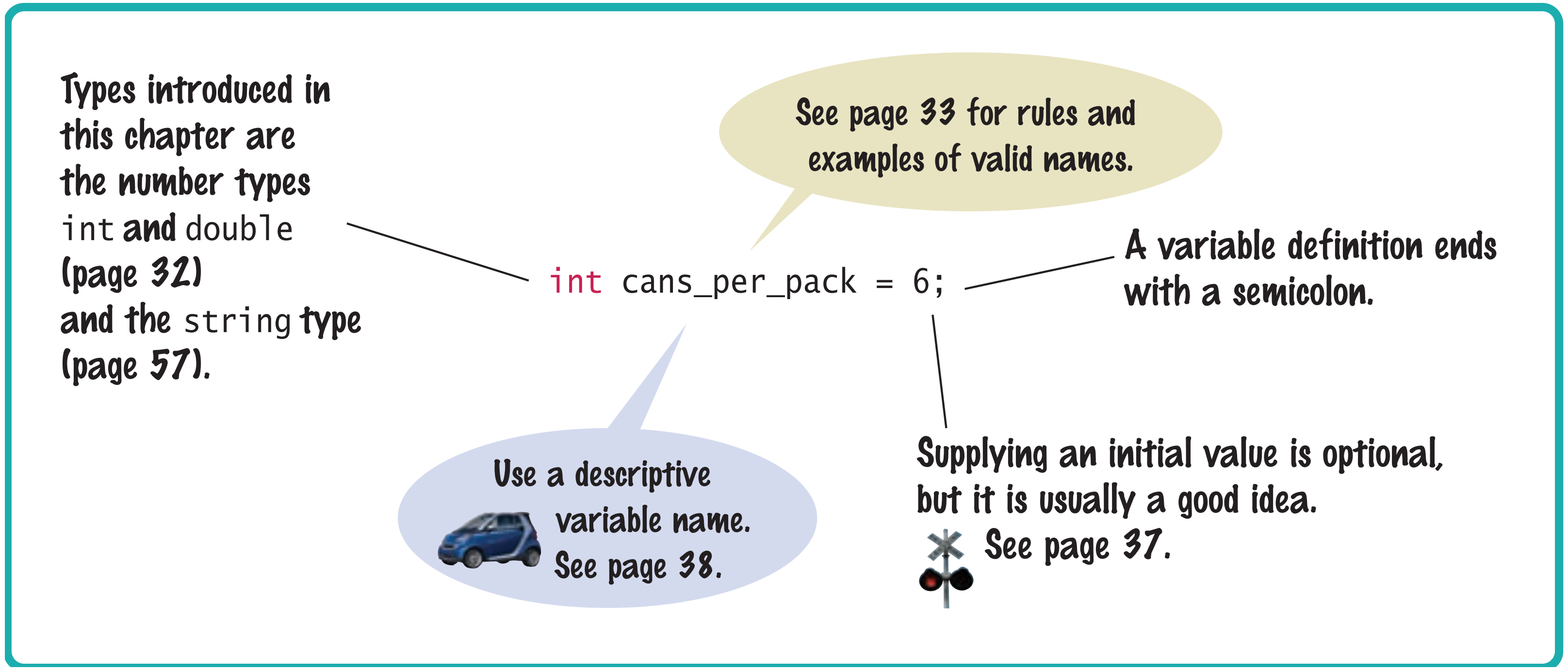
Variable Definitions

- When creating variables, the programmer specifies the **type** of information to be stored.

- A variable is often given an initial value.
    - ***Initialization*** is putting a value into a variable when the variable is created.
    - Initialization is **not required**.

Variables: chunks of memory


alright! what do they want in it?
Terrance, they want another 64 to store an inneger!
Nuthin! They didn't say! Just leave whatever was left in from before - you don't wanna hurt your back flipping switches you don't need to



```
int cans_per_pack = 6;
```

Types introduced in this chapter are the number types `int` and `double` (page 32) and the `string` type (page 57).

See page 33 for rules and examples of valid names.

A variable definition ends with a semicolon.

Use a descriptive variable name. See page 38.

Supplying an initial value is optional, but it is usually a good idea. See page 37.

Variable Definitions

Variable Definitions: example

The following statement defines a variable:

```
int cans_per_pack = 6;
```

`cans_per_pack` is the variable's name.

`int` indicates that the variable **cans_per_pack** will hold integers. Other variable types covered later will hold *strings* and *floating-point numbers*.

`= 6` indicates that the variable `cans_per_pack` will initially contain the value 6.

Like all statements, it must end with a semicolon.

To the CodeMobile

Variable Definitions: more examples

Table 1: Variable Definitions in C++

	Comment
int cans = 6;	Defines an integer variable and initializes it with 6.
int total = cans + bottles;	The initial value need not be a constant. (Of course, cans and bottles must have been previously defined.)
int bottles = "10";	Error: You cannot initialize an int variable with a string.
int bottles;	Defines an integer variable without initializing it. This can be a cause for errors—see Common Error 2.2.
int cans, bottles;	Defines two integer variables in a single statement. In this book, we will define each variable in a separate statement.
bottles = 1;	Caution: The type is missing. This statement is not a definition but an assignment of a new value to an existing variable—see Section 2.1.4.

Table 2: Number Literals

	Type	Comment
6	int	An integer has no fractional part.
–6	int	Integers can be negative.
0	int	Zero is an integer.
0.5	double	A number with a fractional part has type double.
1.0	double	An integer with a fractional part .0 has type double.
1E6	double	A number in exponential notation: 1 × 106 or 1000000. Numbers in exponential notation always have type double.
2.96E-2	double	Negative exponent: 2.96 × 10–2 = 2.96 / 100 = 0.0296
100,000		Error: Do not use a comma as a decimal separator.
3 1/2		Error: Do not use fractions; use decimal notation: 3.5.

Table 3: Variable Names

Variable Name	Comment
can_volume1	Variable names consist of letters, numbers, and the underscore character.
x	In mathematics, you use short variable names such as x or y. This is legal in C++, but not very common, because it can make programs harder to understand (see Programming Tip 2.1)
Can_volume	Caution: Variable names are case sensitive. This variable name is different from can_volume.
6pack	Error: Variable names cannot start with a number.
can volume	Error: Variable names cannot contain spaces.
double	Error: You cannot use a reserved word as a variable name.
ltr/fl.oz	Error: You cannot use symbols such as . or /

The Assignment Statement

- The contents in variables can “vary” over time (hence the name!).
- Variables can be changed by
    - assigning to them
        - **The assignment statement** (”=“)
    - using the increment or decrement operator (++, --)
    - inputting into them
        - The input statement (“cin”)

Assignment Statement Example

- An *assignment statement* stores a new value in a variable, replacing the previously stored value.

```
cans_per_pack = 8;
```

- This assignment statement changes the value stored in **`cans_per_pack`** to be 8.

- The previous value is replaced.

Assignment Statement: defining vs. assigning

- There is an important difference between a variable definition and an assignment statement:

```
int cans_per_pack = 6; // Variable definition
...
cans_per_pack = 8; // Assignment statement
```

- The first statement is the *definition* of **`cans_per_pack`**.
- The second statement is an *assignment statement.*
  - An *existing* variable's contents are replaced.
- A variable's definition must occur ***only once*** in a program. The same variable may be in several assignment statements in a program.

The Meaning of the Assignment = Symbol

- The = in an assignment does ***not*** mean the left hand side is equal to the right hand side as it does in math.
- = is an instruction to do something:

  copy the value of the expression on the right
  into the variable on the left.

- Consider what it would mean, mathematically, to state:

```
counter = counter + 2;
```

counter *EQUALS* counter + 2

Assignment Examples

```
counter = 11; // set counter to 11
counter = counter + 2; // increment
```

1. First statement assigns 11 to counter
2. Second statement looks up what is currently in the variable counter (11)
3. Then it adds 2 and copies the result of the addition into the variable on the left, changing counter to 13

Constants

- Sometimes the programmer knows certain values just from analyzing the problem
  - For this kind of information, use the reserved word **`const`**.
- The reserved word **`const`** is used to define a constant.
- A **`const`** is a "variable" whose contents cannot be changed and must be set when created.
  (Most programmers just call them constants, not variables.)
- Constants are commonly written using capital letters to distinguish them visually from regular variables:

```
const double BOTTLE_VOLUME = 2;
```

Constants Prevent Unclear Numbers in Code

Another good reason for using constants:

```
    double volume = bottles * 2;
```

What does that 2 mean?

If we use a constant there is no question:

```
    double volume = bottles * BOTTLE_VOLUME;
```

Constants Prevent Unclear Numbers in Code (2)

And still another good reason for using constants:

```
double bottle_volume = bottles * 2;
double can_volume = cans * 2;
```

What does *that* 2 mean?

— *WHICH 2?*

It is not good programming practice to use magic numbers.

Use **constants**.

Constants Prevent Unclear Numbers in Code (3)

And it can get even worse …

Suppose that the number 2 appears hundreds of times throughout a five-hundred-line program?

Now we need to change the BOTTLE_VOLUME to 2.23 (because we are now using a bottle with a different shape)

How to change ***only*** some of those 2’s?

Constants again

Constants to the rescue!

```
const double BOTTLE_VOLUME = 2.23;
const double CAN_VOLUME = 2;
...
double bottle_volume = bottles * BOTTLE_VOLUME;
double can_volume = cans * CAN_VOLUME;
```

Comments

- *Comments* are explanations for human readers of your code (other programmers or your instructor).
- The compiler ignores comments completely.
- A leading double slash // tells the compiler the remainder of this line is a comment, to be ignored
- For example,

```
double can_volume = 0.355; // Liters in a 12-ounce can
```

Comments: // or /* multi-line */

Comments can be written in two styles:

- Single line:

```
double can_volume = 0.355; // Liters in a 12-ounce can
```

The compiler ignores everything after // to the end of line

- Multiline for longer comments, where the compiler ignores everything between /* and */

```
 /*
   This program computes the volume (in liters)
   of a six-pack of soda cans.
*/
```

Common Error: Using Undefined Variables

You must define a variable before you use it for the first time.

For example, the following sequence of statements would not be legal:

```
double can_volume = 12 * liter_per_ounce;
double liter_per_ounce = 0.0296;
```

Statements are compiled in top to bottom order.

When the compiler reaches the first statement, it does not know that **`liter_per_ounce`** will be defined in the next line, and it reports an error.

Common Error: Using Uninitialized Variables

- Initializing a variable is not required, but there is always a value in every variable, even uninitialized ones.
- Some value will be there, left over from some previous calculation or simply the random value there when the transistors in RAM were first turned on.

```
    int bottles; // Forgot to initialize
    int bottle_volume = bottles * 2;
```

What value would be output from the following statement?

```
cout << bottle_volume << endl;
```

Errors!

Common Error – Omitting Semicolons errors

Omitting a semicolon (or two), in this case at the end of the `cout` statement

```
#include <iostream>

using namespace std;
int main()
{
      cout << "Hello, World!" << endl
      return 0;
}
```

Syntax errors

Without that semicolon you actually wrote:

```
cout << "Hello, World!" << endl return 0;
```

which thoroughly confuses the compiler with the `endl` immediately followed by the `return`!

- This is a *compile-time error* or *syntax error*.
- A syntax error is a part of a program that does not conform to the rules of the programming language.

Errors: Misspellings

- Suppose you (accidentally of course) wrote:

```
cot << "Hello World!" << endl;
```

- This will cause a compile-time error and the compiler will complain that it has no clue what you mean by cot.
- The exact wording of the error message is dependent on the compiler, but it might be something like

“Undefined symbol cot” or "Unknown identifier".

How many errors?

- The compiler will not stop compiling, and will most likely list lots and lots of errors that are caused by the first one it encountered.
- You should fix only those error messages that make sense to you, **starting with the first one**, and then recompile (after SAVING, of course!).

Logic Errors

Consider this:

```
cout << "Hollo, World!" << endl;
```

- *Logic errors* or *run-time errors* are errors in a program that compiles (the syntax is correct), but executes without performing the intended action.
- The programmer must thoroughly inspect and test the program to guard against logic errors.
    - *Testing and repairing a program usually takes more time than writing it in the first place, but is essential !*

Errors: Run-Time Exceptions

Some kinds of run-time errors are so severe that they generate an *exception*: a signal from the processor that aborts the program with an error message.

For example, if your program includes the statement

```
  cout << 1 / 0;
```

Your program may terminate with a “divide by zero” exception.

Errors: extra or misspelled main() function

- Every C++ program must have one and only one **`main`** function.

- Most C++ programs contain other functions besides **`main`** (more about functions next week).

Errors: C++ is Case Sensitive

C++ is *case sensitive.* Typing:

```
        int Main()
```

will compile but will not link.

A link-time error occurs here when the linker cannot find the **`main`** function – because you did not define a function named **`main`**. (**`Main`** is fine as a name but it is not the same as **`main`** and there has to be one **`main`** somewhere.)

If you want to learn more about the build process, read this. The content in this webpage is not a part of the syllabus and will not be on any course related assignments.

Making your Program Readable (by Humans)

C++ has *free-form layout*

```
int main(){cout<<"Hello, World!"<<endl;return 0;}
```

- *will* compile (but is practically impossible to read)

A good program is readable:

- code spaced across multiple lines, one statement per line
- follows indentation conventions