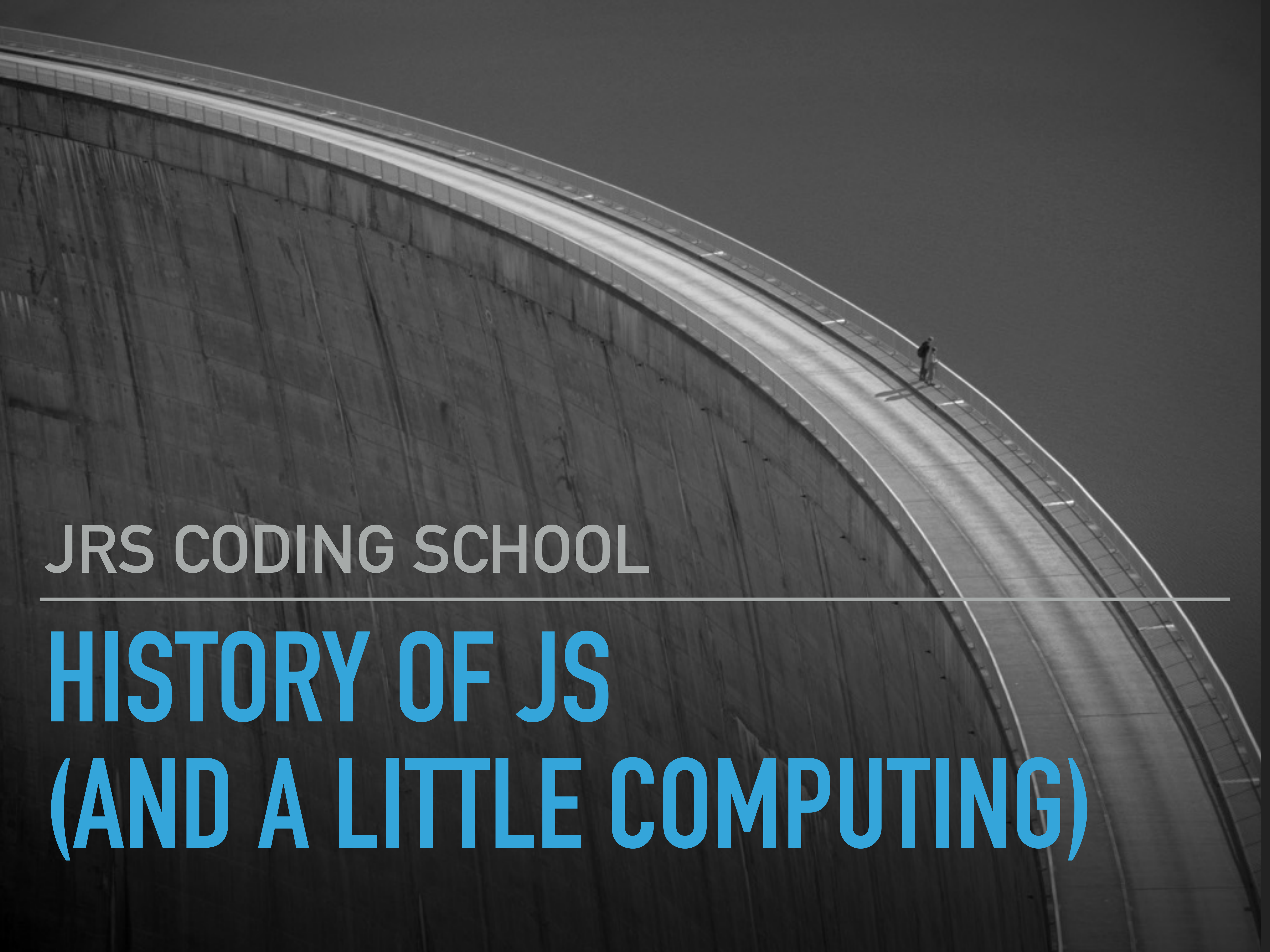


JRS CODING SCHOOL

# HISTORY OF JS (AND A LITTLE COMPUTING)

## HOW DO HUMANS INTERACT WITH COMPUTERS

- 2 techniques to making computers understand us:
  - Build interfaces that mimic the physical world.  These are limiting.  Point and click only goes so far.
  - Instructing the computer to perform arbitrary tasks, we've had more luck with an approach that makes use of our talent for language: teaching the machine a language
- We can combine words and phrases, nouns and verbs, to instruct the computer what to do.

## WHAT IS A PROGRAM

- ”It is a piece of text typed by a programmer”
- “it is the directing force that makes the computer do what it does”
- “it is data in the computer’s memory, yet it controls the actions performed on this same memory.”
- “Its an immaterial (not physical) machine“
- “A program is a building of thought.”
- “Without care, a program’s size and complexity will grow out of control, confusing even the person who created it. Keeping programs under control is the main problem of programming.”

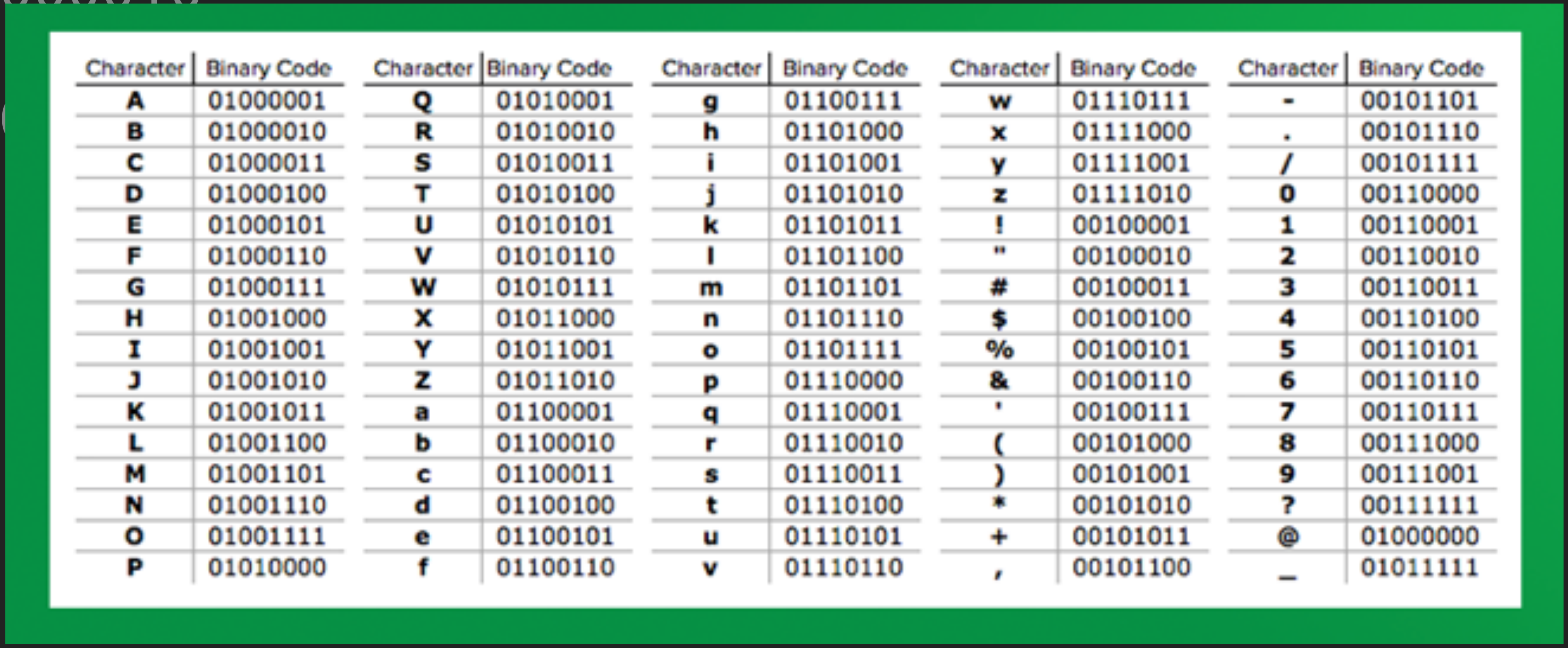
## WHAT IS A COMPUTER

- "A computer is a machine built to act as a host for these immaterial machines. "
- Computers do simple things at an incredibly high speed.
- "A program can ingeniously combine an enormous number of these simple actions in order to do very complicated things."

## IN THE BEGINNING…

- In the beginning, at the birth of computing, there were no programming languages. Programs looked something like this:

- 00110001 00000000 00000000 — 1. Store the number 0 in memory location 0.
- 00110001 00000001 00000001 — 2. Store the number 1 in memory location 1.
- 00110011 00000001 00000010 — 3. Store the value of memory location 1 in memory location 2.
- 01010001 00001011 00000010
- 00100010 00000010 000



| Character | Binary Code | Character | Binary Code | Character | Binary Code | Character | Binary Code | Character | Binary Code |
|---|---|---|---|---|---|---|---|---|---|
| A | 01000001 | Q | 01010001 | g | 01100111 | w | 01110111 | - | 00101101 |
| B | 01000010 | R | 01010010 | h | 01101000 | x | 01111000 | . | 00101110 |
| C | 01000011 | S | 01010011 | i | 01101001 | y | 01111001 | / | 00101111 |
| D | 01000100 | T | 01010100 | j | 01101010 | z | 01111010 | 0 | 00110000 |
| E | 01000101 | U | 01010101 | k | 01101011 | ! | 00100001 | 1 | 00110001 |
| F | 01000110 | V | 01010110 | l | 01101100 | " | 00100010 | 2 | 00110010 |
| G | 01000111 | W | 01010111 | m | 01101101 | # | 00100011 | 3 | 00110011 |
| H | 01001000 | X | 01011000 | n | 01101110 | $ | 00100100 | 4 | 00110100 |
| I | 01001001 | Y | 01011001 | o | 01101111 | % | 00100101 | 5 | 00110101 |
| J | 01001010 | Z | 01011010 | p | 01110000 | & | 00100110 | 6 | 00110110 |
| K | 01001011 | a | 01100001 | q | 01110001 | ' | 00100111 | 7 | 00110111 |
| L | 01001100 | b | 01100010 | r | 01110010 | ( | 00101000 | 8 | 00111000 |
| M | 01001101 | c | 01100011 | s | 01110011 | ) | 00101001 | 9 | 00111001 |
| N | 01001110 | d | 01100100 | t | 01110100 | * | 00101010 | ? | 00111111 |
| O | 01001111 | e | 01100101 | u | 01110101 | + | 00101011 | @ | 01000000 |
| P | 01010000 | f | 01100110 | v | 01110110 | , | 00101100 | _ | 01011111 |

## IMPERATIVE COMMANDS

- To program early computers, it was necessary to set large arrays of switches in the right position or punch holes in strips of cardboard and feed them to the computer.
- Tedious and error-prone!
- Telling the computer in detail *how* to do something.
- imperative programming is a programming paradigm that uses statements that tell a computer *how* to change a program's state.

- Telling the computer in detail *how* to do something.
  - 1. Store the number 0 in memory location 0.
  - 2. Store the number 1 in memory location 1.
  - 3. Store the value of memory location 1 in memory location 2.
  - 4. Subtract the number 11 from the value in memory location 2.
  - 5. If the value in memory location 2 is the number 0,
  - continue with instruction 9.
  - 6. Add the value of memory location 1 to memory location 0.
  - 7. Add the number 1 to the value of memory location 1.
  - 8. Continue with instruction 3.
  - 9. Output the value of memory location 0.

## IMPERATIVE VS. DECLARATIVE

- Imperative is code that emphasizes *how* to accomplish a task.
- 1'S and 0'S on punch cards is the most imperative way.
  - switch this bit from 1 to 0. The computer doesn't interpret anything.
- Then we started adding programming languages
  - Assembly language, for example. Slightly less imperative.
- Programming languages create a layer of abstraction which makes things more friendly to humans and the code could be interpreted by the computer.
- Languages were added with libraries and patterns and frameworks… all in the goal of making life easier for humans.
- To abstract away more from the human of how the computer gets things done.

## THE SAME PROGRAM AS JAVASCRIPT

- JavaScript offers some improvements
- The **`while`** keyword handles jumping back and forth
- A lot of the *uninteresting detail* of ***how*** to get the job done is removed.

```
var total = 0, count = 1;
while (count <= 10) {
  total += count;
  count += 1;
}
console.log(total);  // → 55
```

## BUT STILL ALL THIS LOOPING AND KEEPING UP WITH STATE IS STILL KINDA IMPERATIVE

- We are still using variables to track state.
- Using loops (uninteresting detail) to tell the computer how to do something.
- Still tedious and error prone.

```
var total = 0, count = 1;
while (count <= 10) {
  total += count;
  count += 1;
}
console.log(total);  // → 55
```

## THERE ARE TWO THINGS WE WANT TO COMPUTER TO DO….

- 1) Get a **range** of numbers from 1 to 10
- 2) **Sum** those numbers
- What are uninteresting details?
  - The counting
  - Loop mechanics
  - Managing state
    - Don't share with with other parts of your program!
    - Dont let it change unexpectedly!

```
var total = 0, count = 1;
while (count <= 10) {
  total += count;
  count += 1;
}
console.log(total);  // → 55
```

## WHAT IF YOU HAD A RANGE FUNCTION AND A SUM FUNCTION

- A range function would give you a list of numbers.
- A sum function would add up a list of numbers.

```
range(1,10)     // [1,2,3,4,5,6,7,8,910]

sum([1,2,3,4,5,6,7,8,910]) // 55

sum(range(1,10)) // 55
```

  - Functions should play a pivotal role in how data is transferred throughout our application.
  - Your code should be organized around this principle.
  - Through this course we will code in the imperative way then evolve to a more functional style. Ex:  sum(range(1,10))  // 55

## JAVASCRIPT

- JavaScript was introduced in 1995
- Originally made to add programs to web pages in the Netscape Navigator browser.
- Been adopted by all other major graphical web browsers.
- Demo:  Developer tools in Chrome
- It has made modern web applications possible–applications with which you can interact directly, without doing a page reload for every action.

## JAVASCRIPT

- Nothing to do with the programming language named Java
- Java was hot at the time.  The name was inspired by marketing considerations.  It was a bad decision.

## ECMASCRIPT STANDARD

- After its adoption outside of Netscape, a standard document was written to describe the way the JavaScript language should work to make sure the various pieces of software that claimed to support JavaScript were actually talking about the same language. This is called the ECMAScript standard, after the Ecma International organization that did the standardization.
- the terms ECMAScript and JavaScript can be used interchangeably

## JAVASCRIPT IS LIBERAL

- Let's don't talk politics in the classroom.  But…
- JavaScript is ridiculously liberal in what it allows.
- Flexibility is bad.  It is very flexible but it doesn't show you where the problems are in a lot of cases.
- Flexibility is good.   It leaves space for a lot of techniques that are impossible in more rigid languages.

## JAVASCRIPT'S RISE TO POWER

- ECMAScript version 3 - JavaScript's ascent to dominance from 2000 to 2010.
- Ambitious improvements to version 4 were abandoned in 2008
- 2009 / Version 5 / ECMAScript 5
  - Much less ambitious version 5 in 2009
  - ECMAScript 5 is fully supported in all modern browsers
- 2015 - ES2015 / Version 6 / ECMAScript 6 /ES6
  - ECMAScript 6 is partially supported in all modern browsers.
- 2016 - ES2016 / Version 7 / ECMAScript 7 /ES7
  - ECMAScript 7 is poorly supported in all browsers.

## JAVASCRIPT CAN BE USED OUTSIDE THE BROWSER

- Some databases, such as MongoDB and CouchDB, use JavaScript as their scripting and query language.
- Several platforms for desktop and server programming, most notably the Node.js project are providing a powerful environment for programming JavaScript outside of the browser.