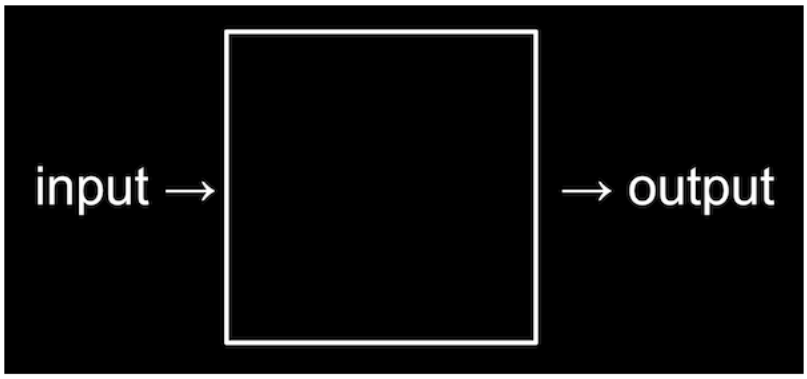
What is computer science?

- Computer science is fundamentally problem-solving.
- We can think of problem-solving as the process of taking some input (details about our problem) and generate some output (the solution to our problem). The "black box" in the middle is computer science.



- We need a way to represent inputs, such that we can store and work with information in a standard way.

Binary

- A computer, at the lowest level, stores data in binary, a numeral system in which there are just two digits, 0 and 1.
- When we first learned to count, we might have used one finger to represent one thing. That system is called unary. When we learned to write numbers with the digits 0 through 9, we learned to use decimal.
- For example, we know the following represents one hundred and twenty-three.

```
1 2 3
```

  - The `3` is in the ones column, the `2` is in the tens column, and the `1` is in the hundreds column.
  - So `123` is 100×1 + 10×2 + 1×3 = 100 + 20 + 3 = 123.
  - Each place for a digit represents a power of ten, since there are ten possible digits for each place.
- In binary, with just two digits, we have powers of two for each place value:

```
4 2 1
0 0 0
```

  - This would still be equal to 0.
- Now if we change the binary value to, say, `0 1 1`, the decimal value would be 3.

```
4 2 1
0 1 1
```

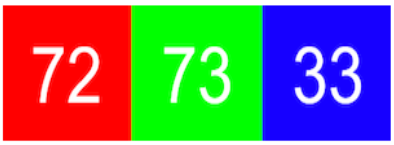
- If we wanted to represent 8, we would need another digit:

```
8 4 2 1
1 0 0 0
```

- And binary makes sense for computers because we power them with electricity, which can be either on or off, so each bit only needs to be on or off. In a computer, there are millions or billions of switches called transistors that can store electricity and represent a bit by being "on" or "off".
- With enough bits, or binary digits, computers can count to any number.
- 8 bits make up one **byte**.

Representing data

- To represent letters, all we need to do is decide how numbers map to letters. Some humans, many years ago, collectively decided on a standard mapping called ASCII. The letter "A", for example, is the number 65, and "B" is 66, and so on. The mapping also includes punctuation and other symbols. Other characters, like letters with accent marks, and emoji, are part of a standard called Unicode that use more bits than ASCII to accommodate all these characters.
  - When we receive an emoji, our computer is actually just receiving a decimal number like `128514` (`11111011000000010` in binary, if you can read that more easily) that it then maps to the image of the emoji.
- An image, too, is comprised of many smaller square dots, or pixels, each of which can be represented in binary with a system called RGB, with values for red, green, and blue light in each pixel. By mixing together different amounts of each color, we can represent millions of colors:



  - The red, green, and blue values are combined to get a light yellow color:

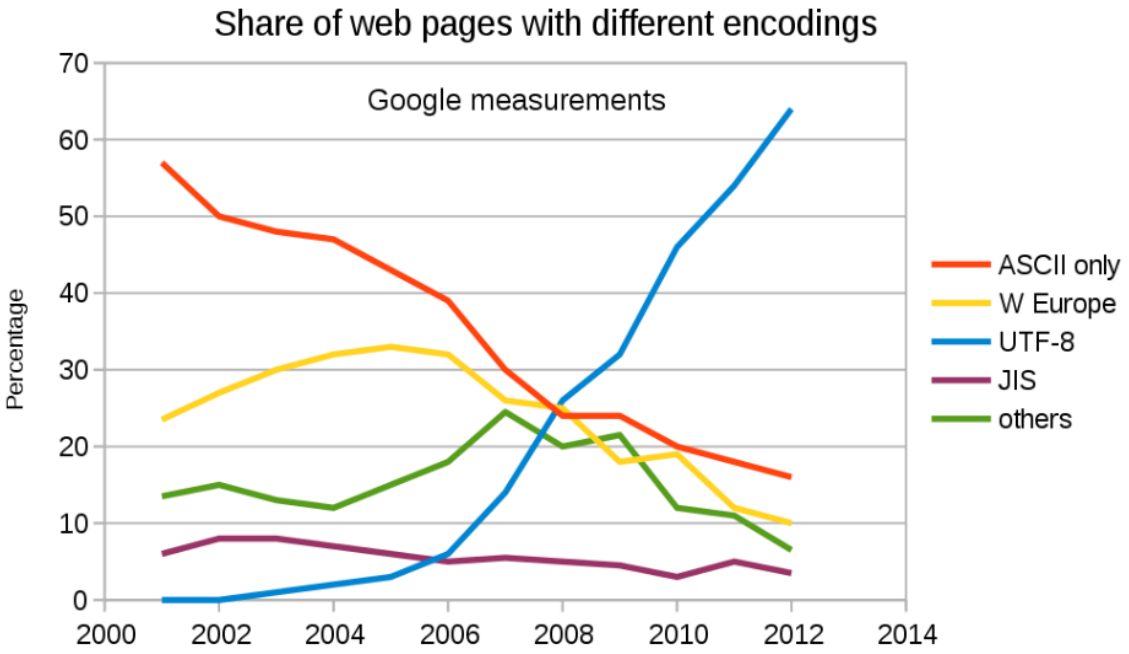
- We can see this in an emoji if we zoom in far enough:

- And computer programs know, based on the context of its code, whether the binary numbers should be interpreted as numbers, or letters, or pixels.
- And videos are just many, many images displayed one after another, at some number of frames per second. Music, too, can be represented by the notes being played, their duration, and their volume.

8-bit codes

Eventually, as 8-, 16- and 32-bit (and later 64-bit) computers began to replace 12-, 18- and 36-bit computers as the norm, it became common to use an 8-bit byte to store each character in memory, providing an opportunity for extended, 8-bit relatives of ASCII.

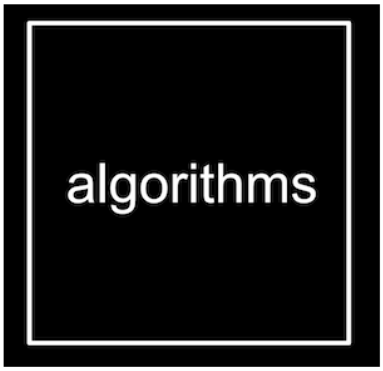
UTF-8 (8-bit Unicode Transformation Format) is capable of encoding all 1,112,064 valid code points in Unicode using one to four one-byte (8-bit) code units. The encoding is defined by the Unicode Standard, and was originally designed by Ken Thompson and Rob Pike. The name is derived from *Unicode* (or *Universal Coded Character Set*) *Transformation Format – 8-bit*.



Usage of the main encodings on the web from 2001 to 2012 as recorded by Google, with UTF-8 overtaking all others in 2008 and over 60% of the web in 2012. The ASCII-only figure includes all web pages that only contain ASCII characters, regardless of the declared header.

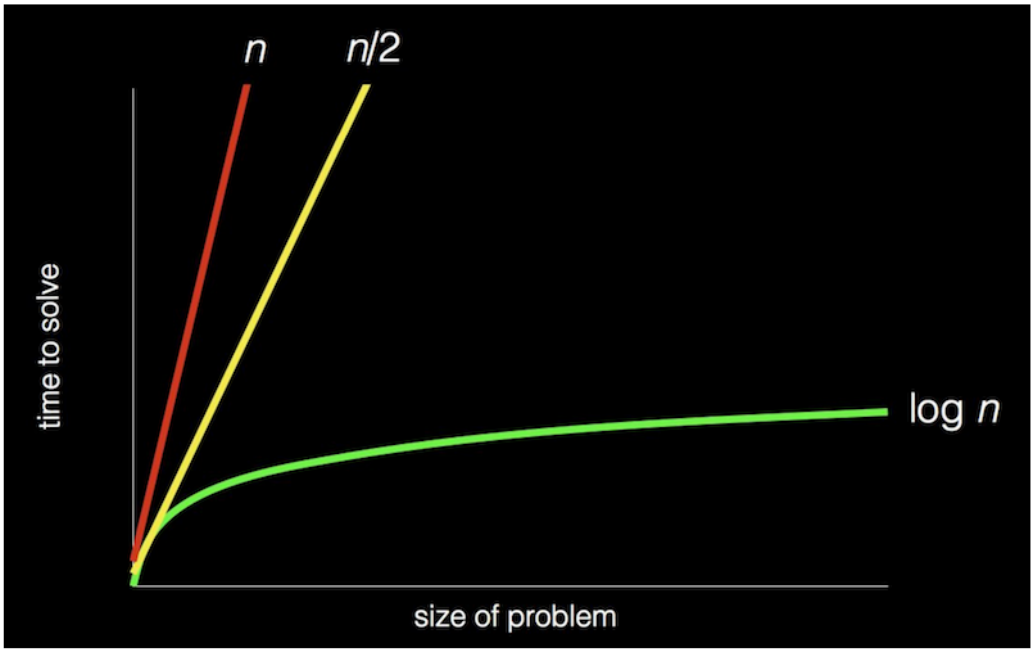
Algorithms

- So now we can represent inputs and outputs. The black box earlier will contain *algorithms*, step-by-step instructions for solving a problem:



- Let's say we wanted to find a friend, Mike Smith, in a phone book.
  - We could start by flipping through the book, one page at a time, until we find Mike Smith or reach the end of the book.
  - We could also flip two pages at a time, but if we go too far, we'll have to know to go back a page.
  - But an even more efficient way would be opening the phone book to the middle, decide whether Mike will be in the left half or right half of the book (because the book is alphabetized), and immediately throw away half of the problem. We can repeat this, dividing the problem in half each time. With 1024 pages to start, we would only need 10 steps of dividing in half before we have just one page remaining to check.

- In fact, we can represent the efficiency of each of those algorithms with a chart:



  - Our first solution, one page at a time, is like the red line: our time to solve increases linearly as the size of the problem increases.
  - The second solution, two pages at a time, is like the yellow line: our slope is less steep, but still linear.
  - Our final solution, is like the green line: logarithmic, since our time to solve rises more and more slowly as the size of the problem increases. In other words, if the phone book went from 1000 to 2000 pages, we would need one more step to find Mike. If the size doubled again from 2000 to 4000 pages, we would still only need one more step.

- We can write *pseudocode*, an informal syntax that is just a more specific version of English (or other human language) that represents our algorithm:

```
1  Pick up phone book
2  Open to middle of phone book
3  Look at page
4  If Smith is on page
5      Call Mike
6  Else if Smith is earlier in book
7      Open to middle of left half of book
8      Go back to line 3
9  Else if Smith is later in book
10     Open to middle of right half of book
11     Go back to line 3
12 Else
13     Quit
```

- Some of these lines start with verbs, or actions. We'll start calling these *functions*:

```
1  Pick up phone book
2  Open to middle of phone book
3  Look at page
4  If Smith is on page
5      Call Mike
6  Else if Smith is earlier in book
7      Open to middle of left half of book
8      Go back to line 3
9  Else if Smith is later in book
10     Open to middle of right half of book
11     Go back to line 3
12 Else
13     Quit
```

- We also have branches that lead to different paths, like forks in the road, which we'll call *conditions*:

```
1  Pick up phone book
2  Open to middle of phone book
3  Look at page
4  If Smith is on page
5      Call Mike
6  Else if Smith is earlier in book
7      Open to middle of left half of book
8      Go back to line 3
9  Else if Smith is later in book
10     Open to middle of right half of book
11     Go back to line 3
12 Else
13     Quit
```

- And the questions that decide where we go are called *Boolean expressions*, which eventually result to a value of true or false:

```
1  Pick up phone book
2  Open to middle of phone book
3  Look at page
4  If Smith is on page
5      Call Mike
6  Else if Smith is earlier in book
7      Open to middle of left half of book
8      Go back to line 3
9  Else if Smith is later in book
10     Open to middle of right half of book
11     Go back to line 3
12 Else
13     Quit
```

- Finally, we have words that lead to cycles, where we can repeat parts of our program, called *loops*:

```
1  Pick up phone book
2  Open to middle of phone book
3  Look at page
4  If Smith is on page
5      Call Mike
6  Else if Smith is earlier in book
7      Open to middle of left half of book
8      Go back to line 3
9  Else if Smith is later in book
10     Open to middle of right half of book
11     Go back to line 3
12 Else
13     Quit
```