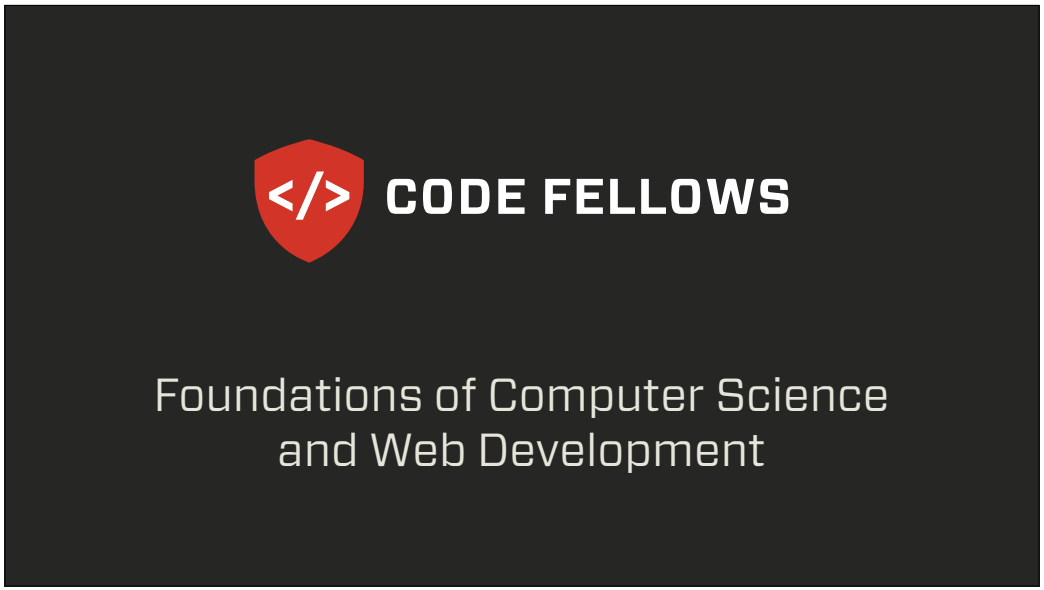
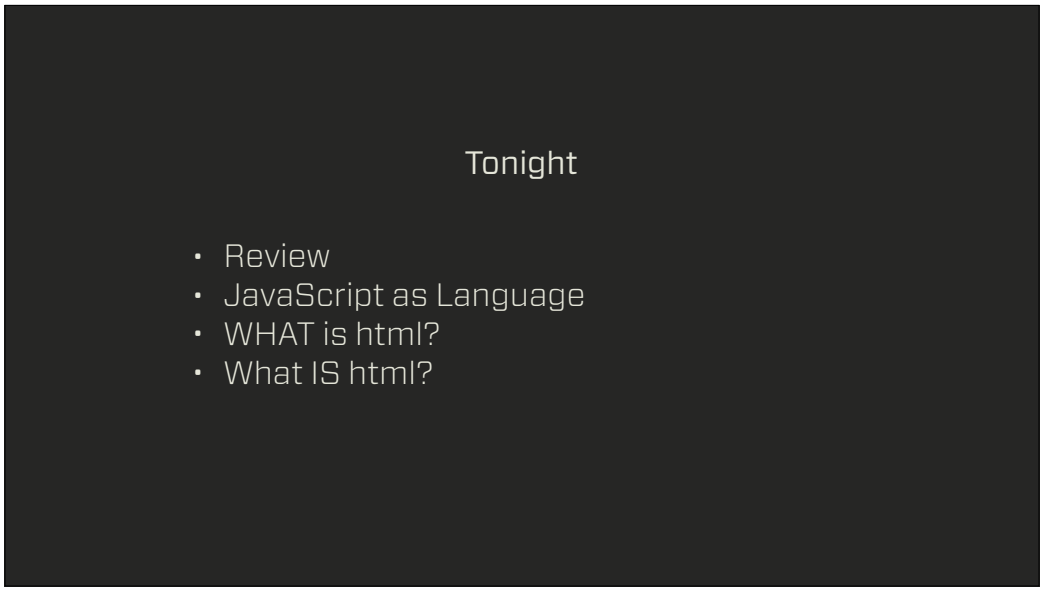
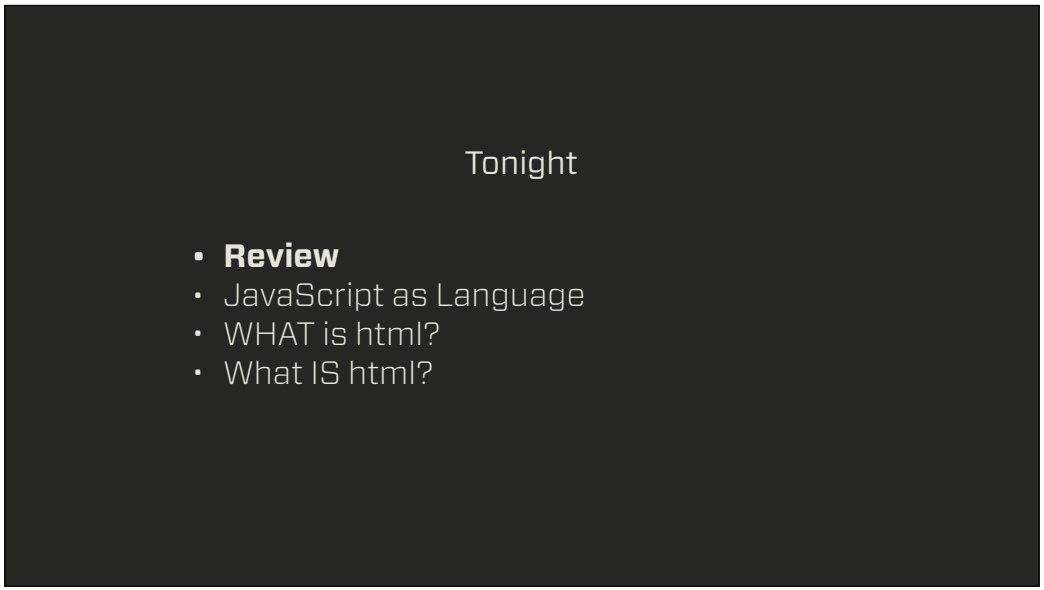


Welcome to class!



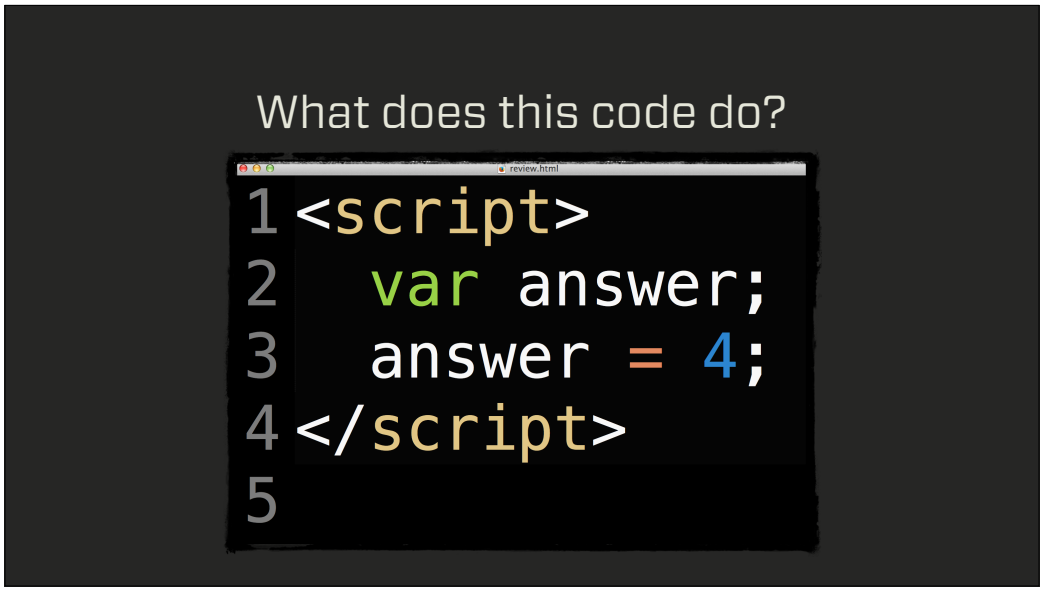
Here’s our plan for tonight.



A quick refresher on some syntax concepts we covered last time…

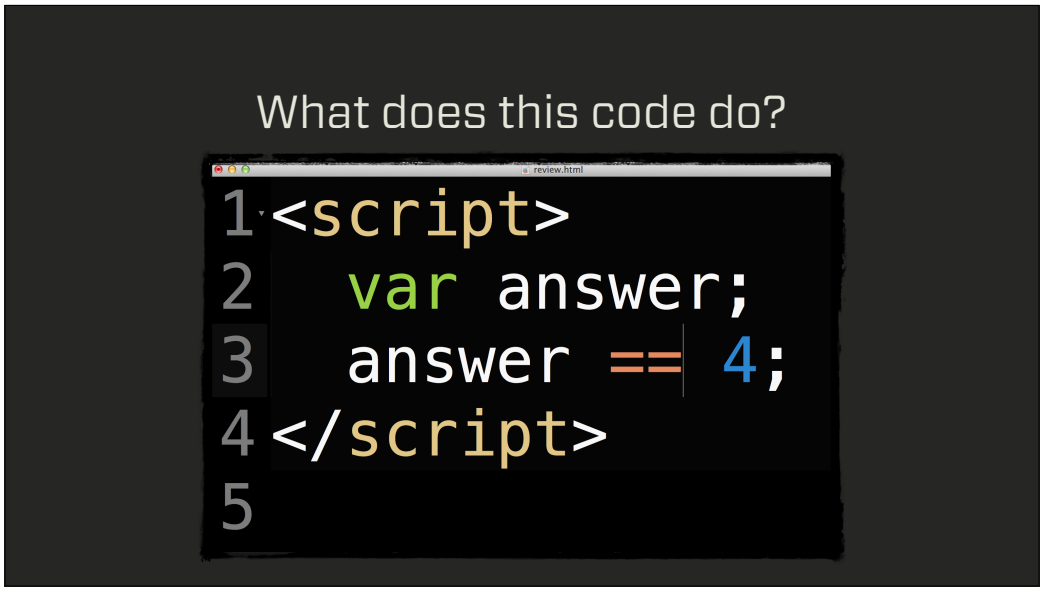
What does this code do?

Let’s start with a little review…

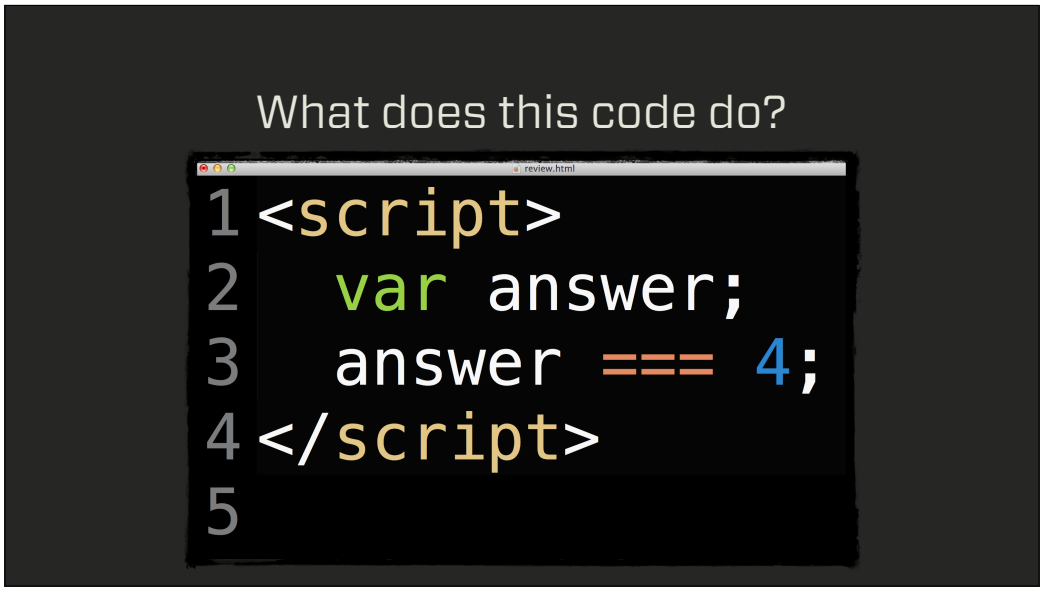


What does this code do? How would you express this in your own words?

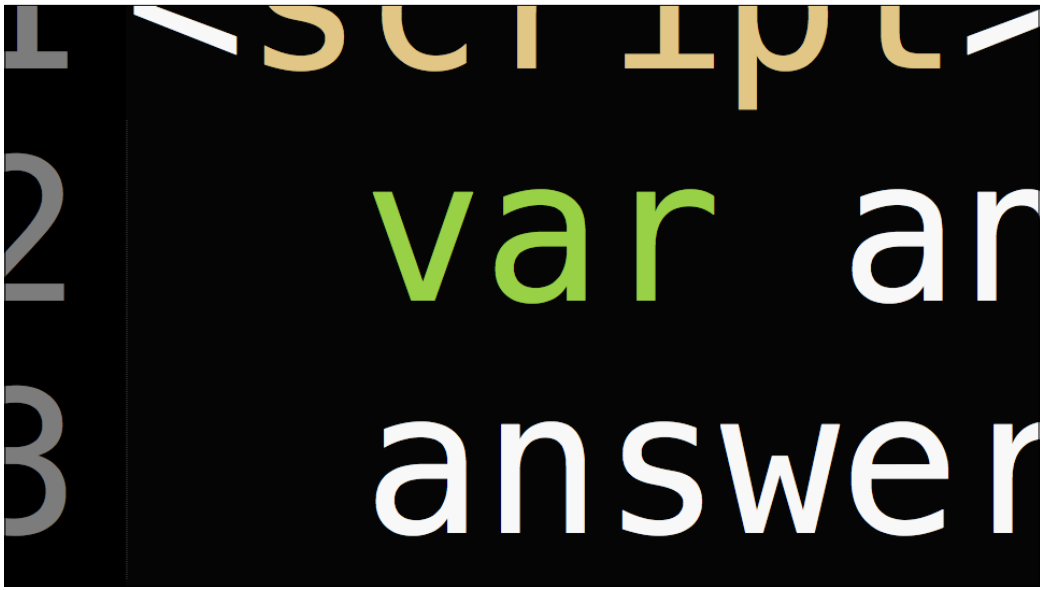
It creates a variable, called “answer”. Line 3 assigns a value to that variable.



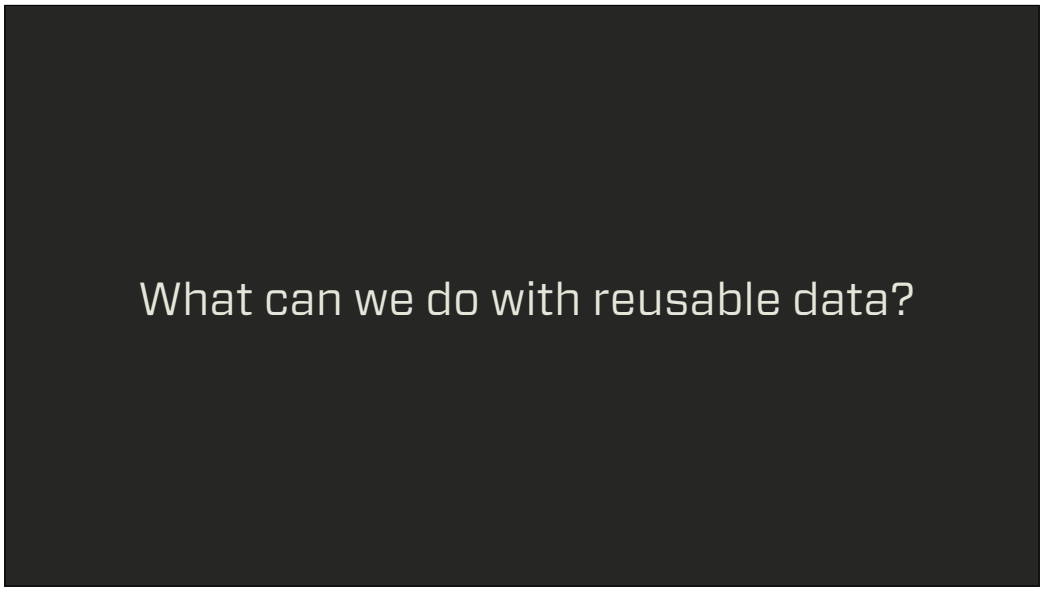
This compares the value in answer and the integer 4 for equality.



This strictly compares the value in answer with the integer 4 for equality.

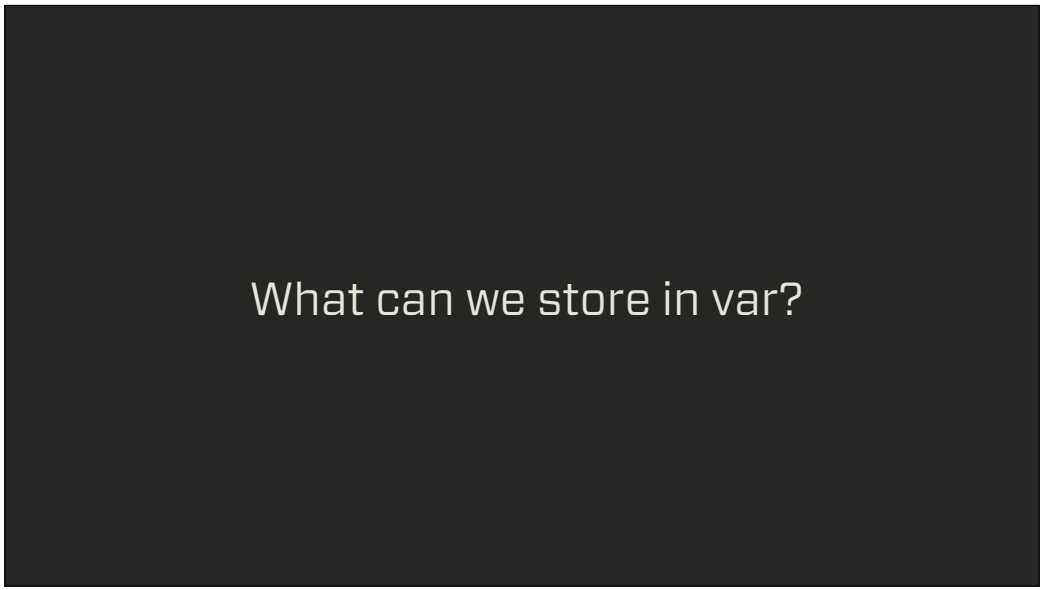


Let’s not overlook the humble var keyword. Using variables in our code allows us to reuse data. Computers are very good at doing things repeatedly. So if we want to repeatedly access some data, and do something with it, we better have a way to store and retrieve it. Var. Reusable data.

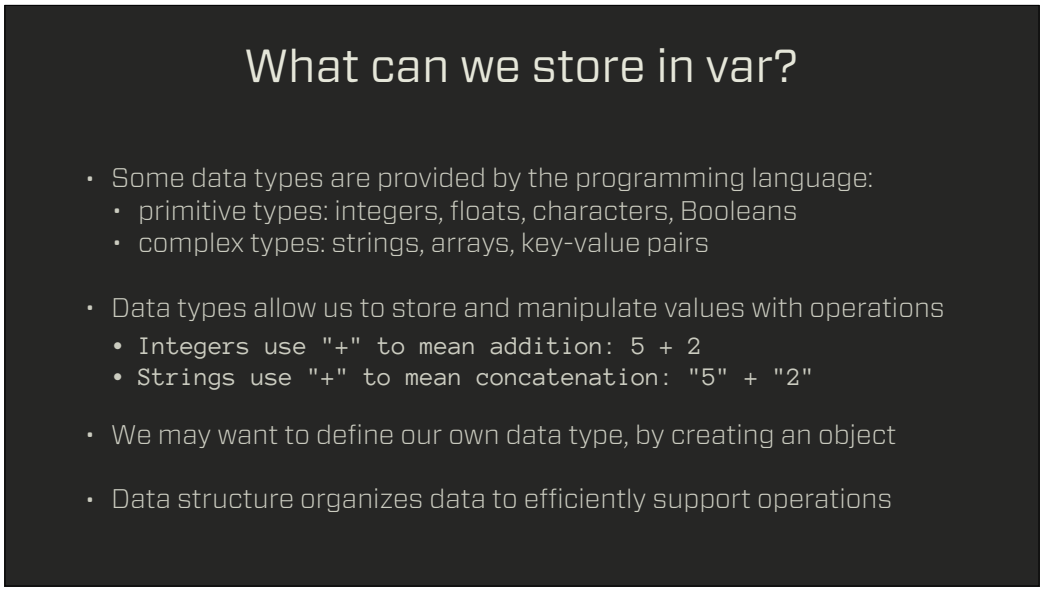


What kinds of things can we do with this reusable data?

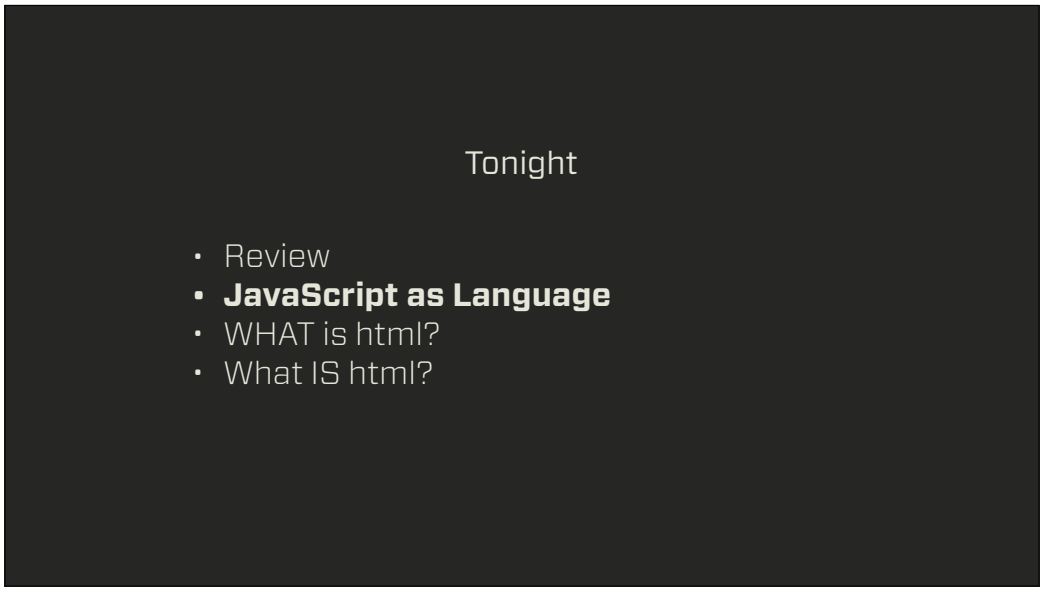
Input, processing, output,



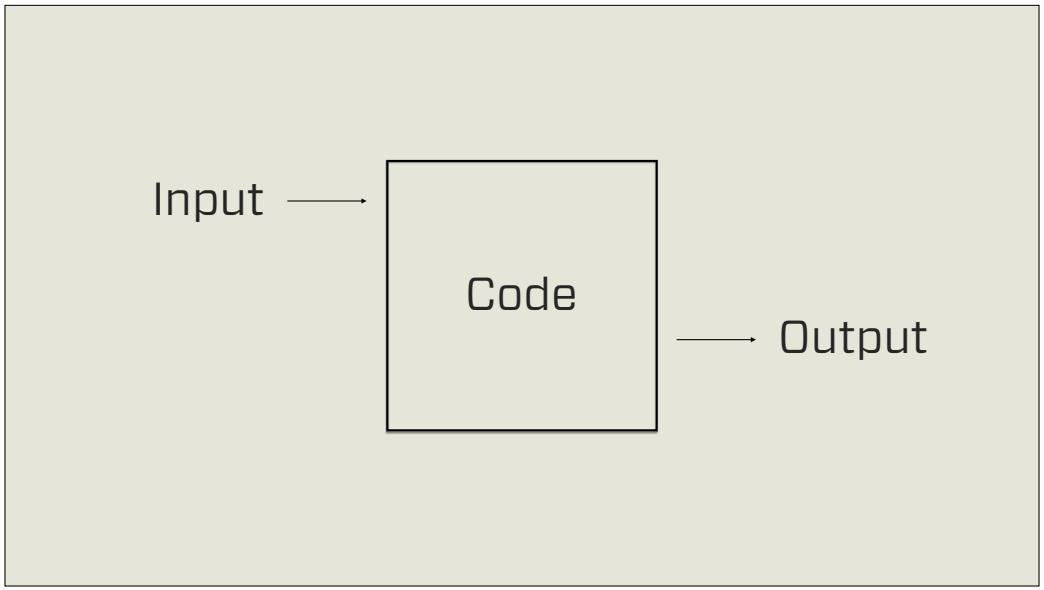
What kind of data can we hold in a var?



What kind of data can we hold in a var? The programming language itself gives us some, and, crucially, allows us to define our own (we'll do so later on).

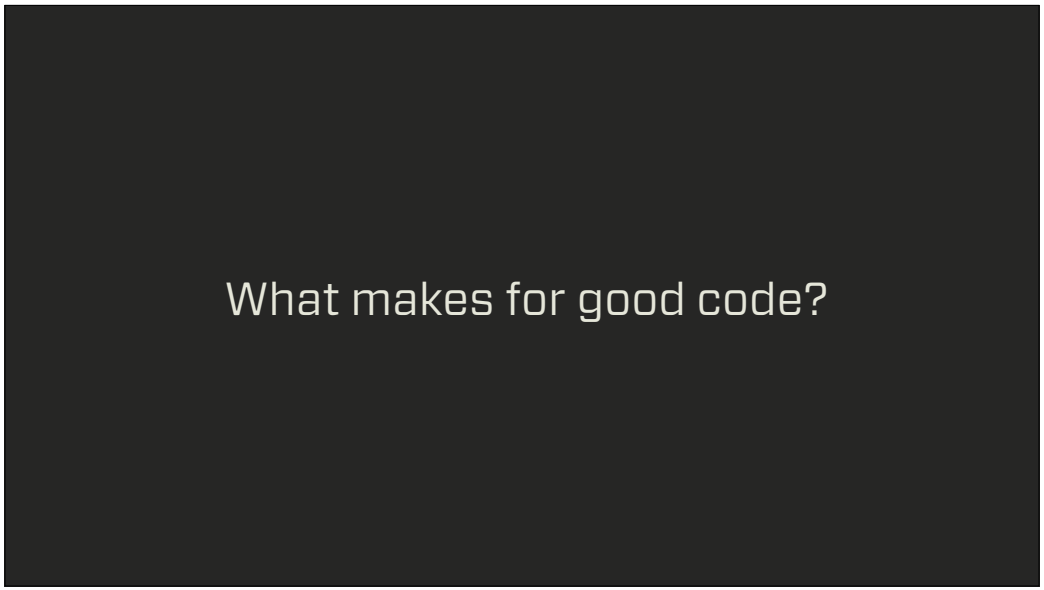


A quick refresher on some syntax concepts we covered last time…

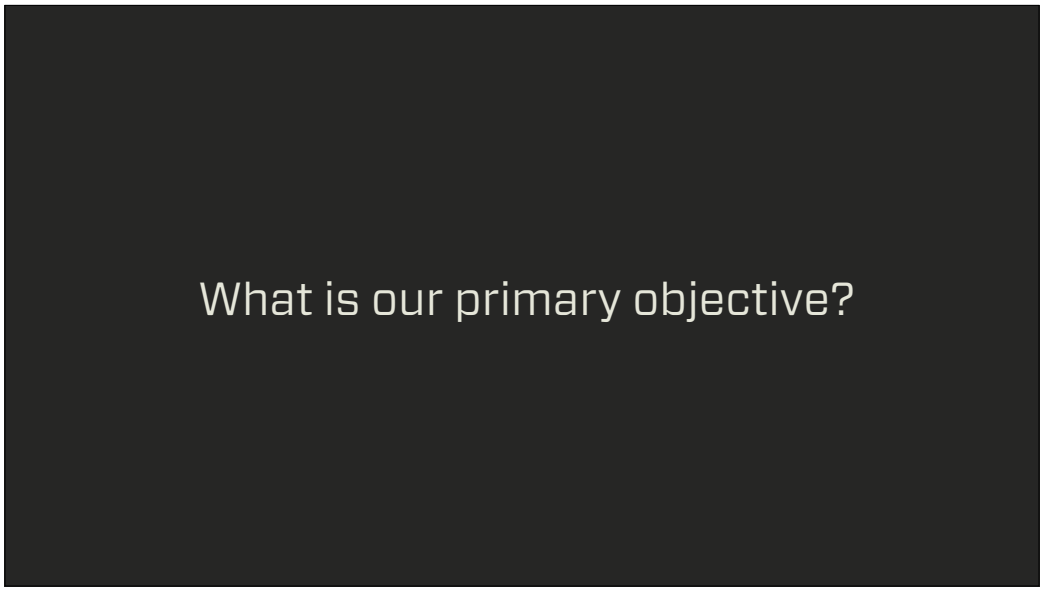


Data is collected from some input (like a user, file, api, database, or hardware sensor).
Data is processed by some sequence of steps.
Data is sent through as output (like a message to a user, a signal to
robot, input to another process, etc).

Potentially many things can happen in that middle Code block. That's where your decision making and problem solving skills come in to play.

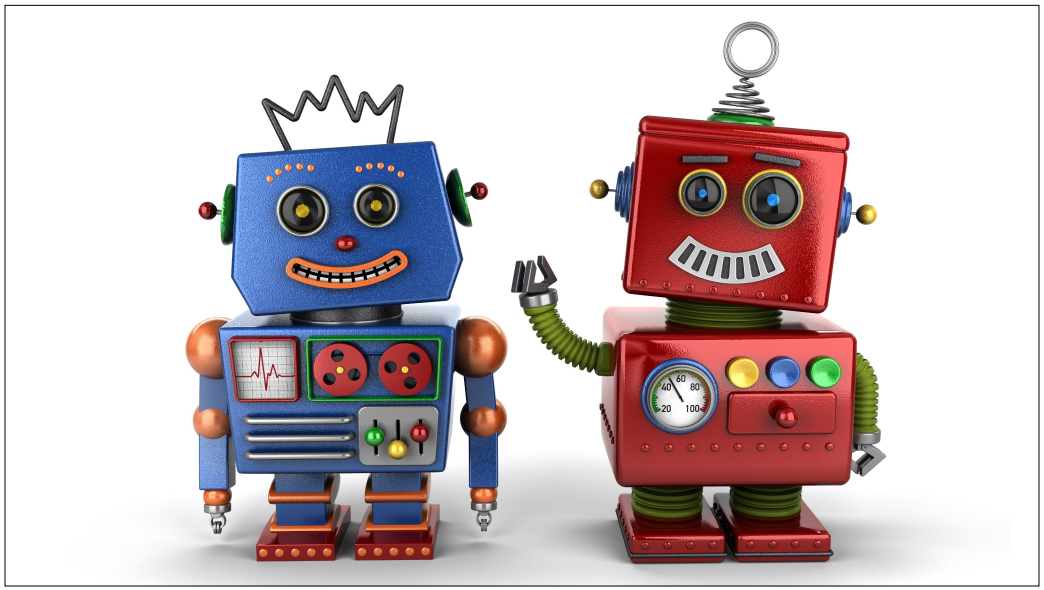


What makes for good code? What should be our highest priority as we write our programs?



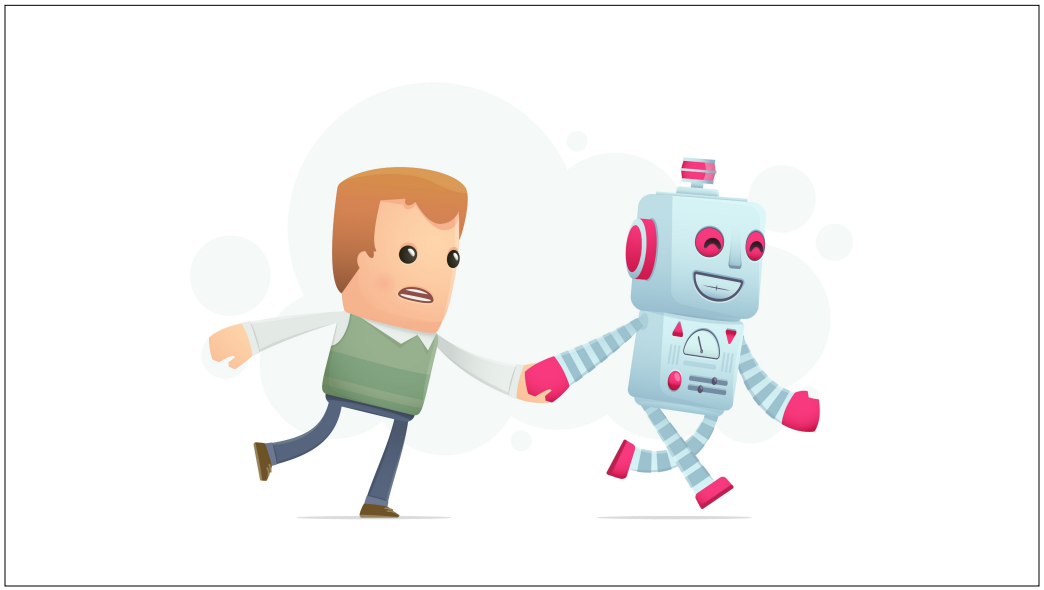
What are we trying to do? Ideas?

We are giving the computer instructions. We want to make the computer do something.



Congrats. You've unlocked your first super-power. You can now talk to the machines. You speak robot.

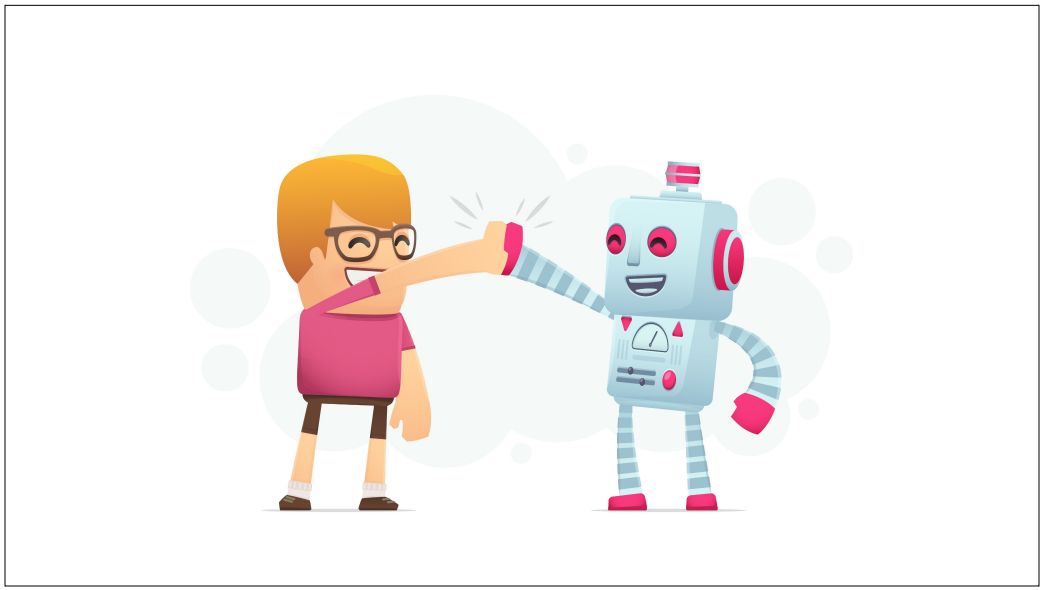
You know how to deliver instructions to your computer. You are learning about how to set those instructions to accomplish your goals.



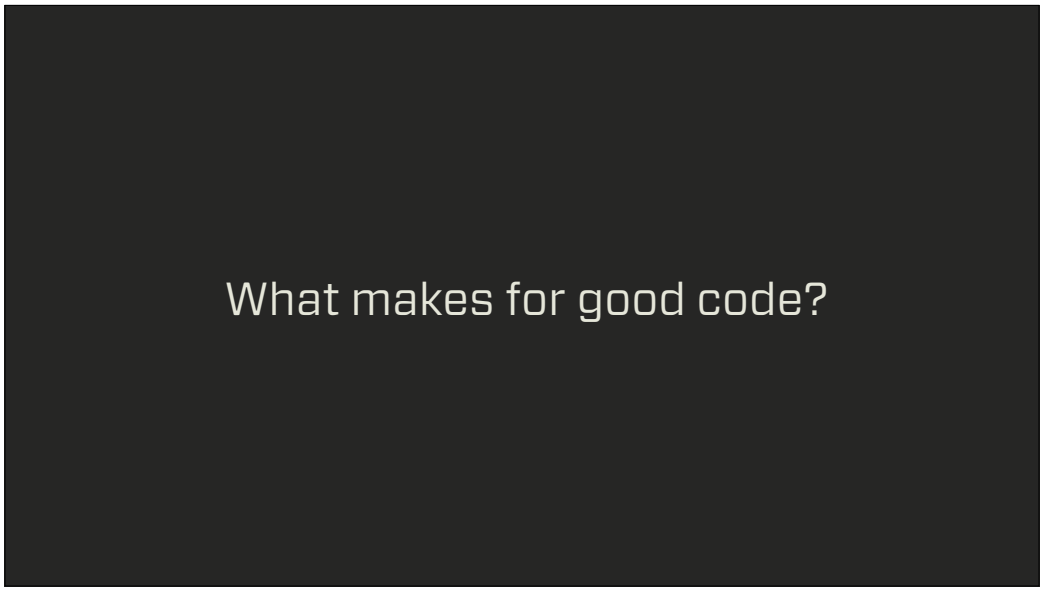
So when the robot uprising comes, and they try to take you away, you will be able to save the day!

But this new language you are learning is not just about telling the computers what to do. We need to keep in mind another objective, nearly as crucial as the first.

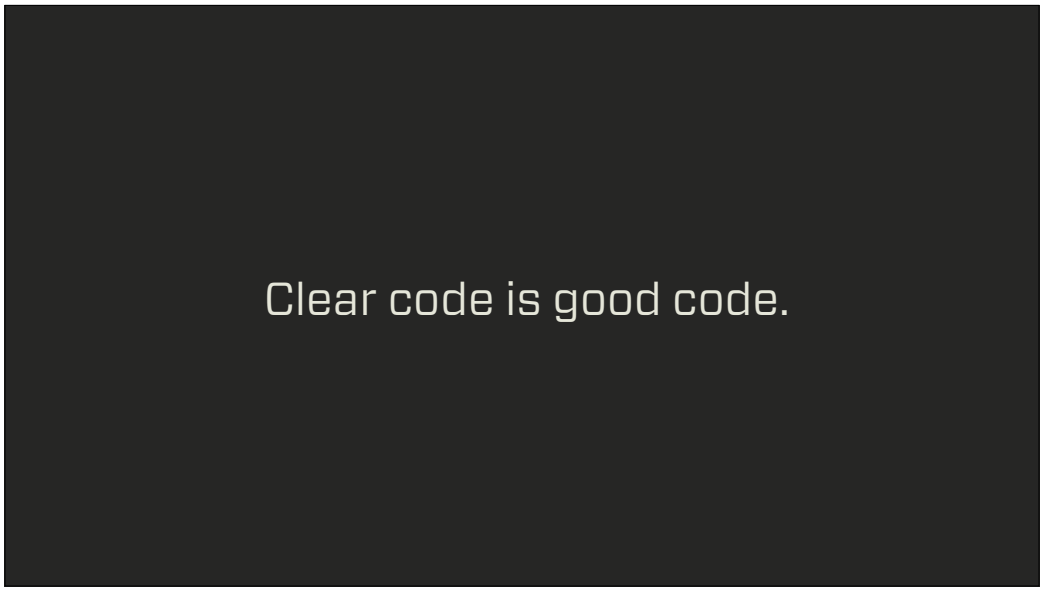
What else do we need to consider when we write code?



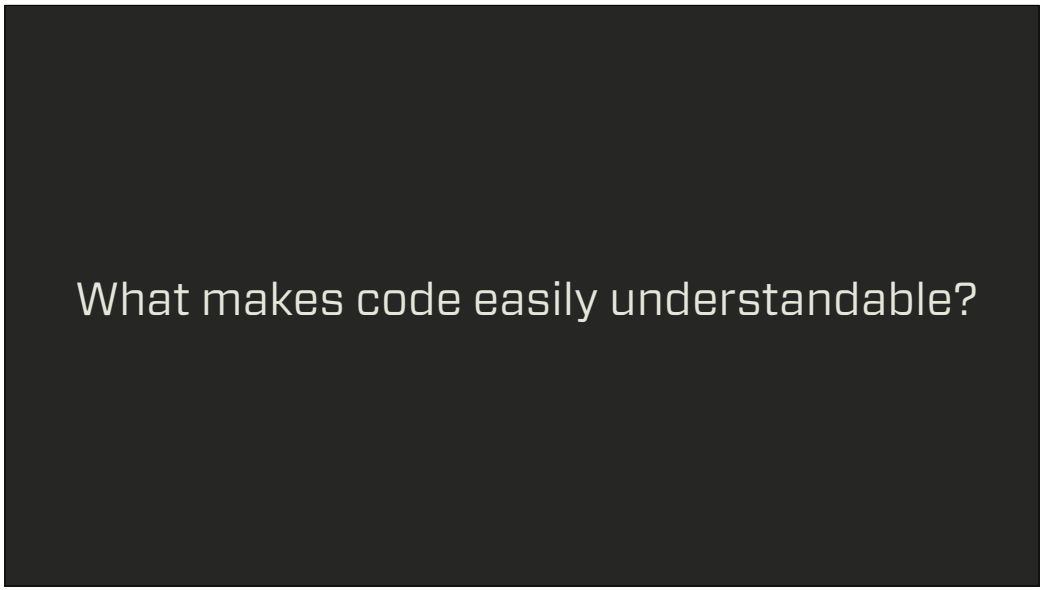
Our code is not one-directional. It’s an interface between people and machines. It needs be written by humans, therefore it needs good human readability.



What makes for good code? What should be our highest priority as we write our programs?



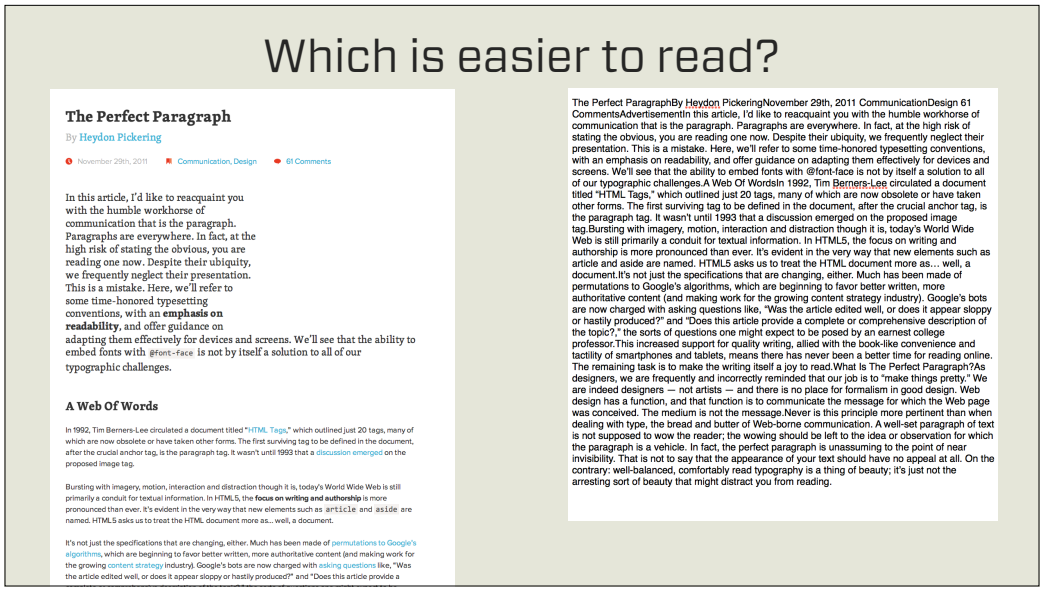
Clarity. Make sure your code is easy to read and understand, above all else.



So what can we do to make code more readable, and easier to understand?

We make careful decisions.

Which is easier to read?



The Perfect Paragraph

By Heydon Pickering

November 29th, 2011 Communication, Design 61 Comments

In this article, I'd like to reacquaint you with the humble workhorse of communication that is the paragraph. Paragraphs are everywhere. In fact, at the high risk of stating the obvious, you are reading one now. Despite their ubiquity, we frequently neglect their presentation. This is a mistake. Here, we'll refer to some time-honored typesetting conventions, with an **emphasis on readability**, and offer guidance on adapting them effectively for devices and screens. We'll see that the ability to embed fonts with `@font-face` is not by itself a solution to all of our typographic challenges.

A Web Of Words

In 1992, Tim Berners-Lee circulated a document titled "HTML Tags," which outlined just 20 tags, many of which are now obsolete or have taken other forms. The first surviving tag to be defined in the document, after the crucial anchor tag, is the paragraph tag. It wasn't until 1993 that a discussion emerged on the proposed image tag.

Bursting with imagery, motion, interaction and distraction though it is, today's World Wide Web is still primarily a conduit for textual information. In HTML5, the **focus on writing and authorship** is more pronounced than ever. It's evident in the very way that new elements such as `article` and `aside` are named. HTML5 asks us to treat the HTML document more as… well, a document.

It's not just the specifications that are changing, either. Much has been made of permutations to Google's algorithms, which are beginning to favor better written, more authoritative content (and making work for the growing content strategy industry). Google's bots are now charged with asking questions like, "Was the article edited well, or does it appear sloppy or hastily produced?" and "Does this article provide a

The Perfect ParagraphBy Heydon PickeringNovember 29th, 2011 CommunicationDesign 61 CommentsAdvertisementIn this article, I'd like to reacquaint you with the humble workhorse of communication that is the paragraph. Paragraphs are everywhere. In fact, at the high risk of stating the obvious, you are reading one now. Despite their ubiquity, we frequently neglect their presentation. This is a mistake. Here, we'll refer to some time-honored typesetting conventions, with an emphasis on readability, and offer guidance on adapting them effectively for devices and screens. We'll see that the ability to embed fonts with @font-face is not by itself a solution to all of our typographic challenges.A Web Of WordsIn 1992, Tim Berners-Lee circulated a document titled "HTML Tags," which outlined just 20 tags, many of which are now obsolete or have taken other forms. The first surviving tag to be defined in the document, after the crucial anchor tag, is the paragraph tag. It wasn't until 1993 that a discussion emerged on the proposed image tag.Bursting with imagery, motion, interaction and distraction though it is, today's World Wide Web is still primarily a conduit for textual information. In HTML5, the focus on writing and authorship is more pronounced than ever. It's evident in the very way that new elements such as article and aside are named. HTML5 asks us to treat the HTML document more as… well, a document.It's not just the specifications that are changing, either. Much has been made of permutations to Google's algorithms, which are beginning to favor better written, more authoritative content (and making work for the growing content strategy industry). Google's bots are now charged with asking questions like, "Was the article edited well, or does it appear sloppy or hastily produced?" and "Does this article provide a complete or comprehensive description of the topic?," the sorts of questions one might expect to be posed by an earnest college professor.This increased support for quality writing, allied with the book-like convenience and tactility of smartphones and tablets, means there has never been a better time for reading online. The remaining task is to make the writing itself a joy to read.What Is The Perfect Paragraph?As designers, we are frequently and incorrectly reminded that our job is to "make things pretty." We are indeed designers — not artists — and there is no place for formalism in good design. Web design has a function, and that function is to communicate the message for which the Web page was conceived. The medium is not the message.Never is this principle more pertinent than when dealing with type, the bread and butter of Web-borne communication. A well-set paragraph of text is not supposed to wow the reader; the wowing should be left to the idea or observation for which the paragraph is a vehicle. In fact, the perfect paragraph is unassuming to the point of near invisibility. That is not to say that the appearance of your text should have no appeal at all. On the contrary: well-balanced, comfortably read typography is a thing of beauty; it's just not the arresting sort of beauty that might distract you from reading.

Why? What attributes does the left have that the right doesn't?

Line breaks, spacing, well-chosen headers, colors & shading, fonts, etc.

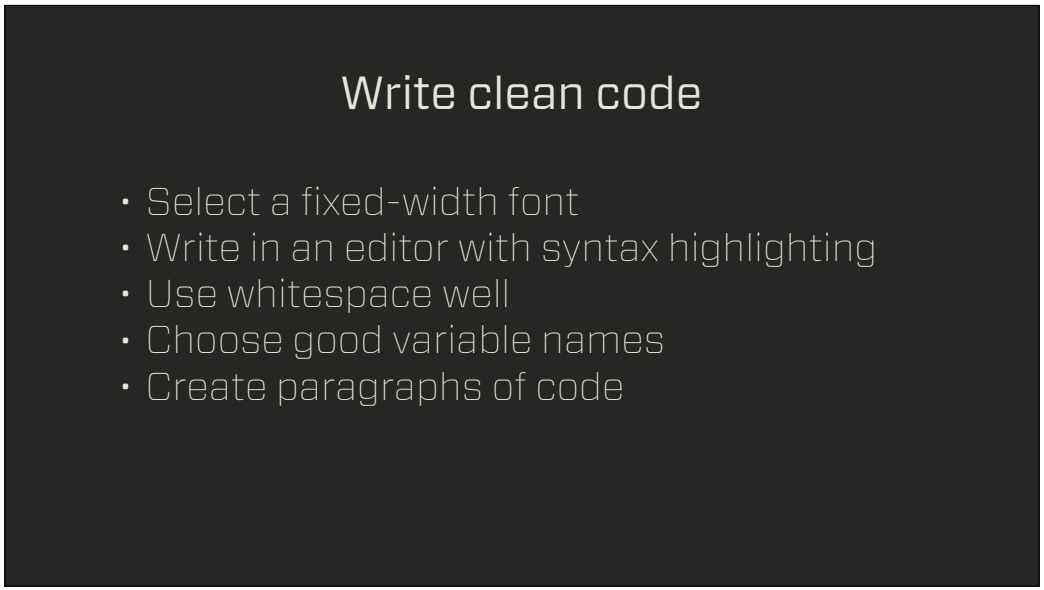
guess-obfuscated.html — f1

```
<script>
eval(function(p,a,c,k,e,d){e=function(c){return c};
  if(!''.replace(/^/,String)){while(c--){d[c]=k[c]
  ||c}k=[function(e){return d[e]}];e=function(){
  return'\\w+'};c=1};while(c--){if(k[c]){p=p.re
  place(new RegExp('\\b'+e(c)+'\\b','g'),k[c])}}
  return p}('8 0,1,2;1=4;0=9("10\'7 12 6?");2=0-1;5
  (0==1){11.13("20, 19 21 22!");3("18 17!")}14{3(
  "15 16: "+2)}',10,23,'guess|answer|difference|ale
rt||if|number|s|var|prompt|What|console|my|log|else
|Off|by|win|You|they|Dagnabit|got|me'.split('|'),0,
  {}))
</script>

```

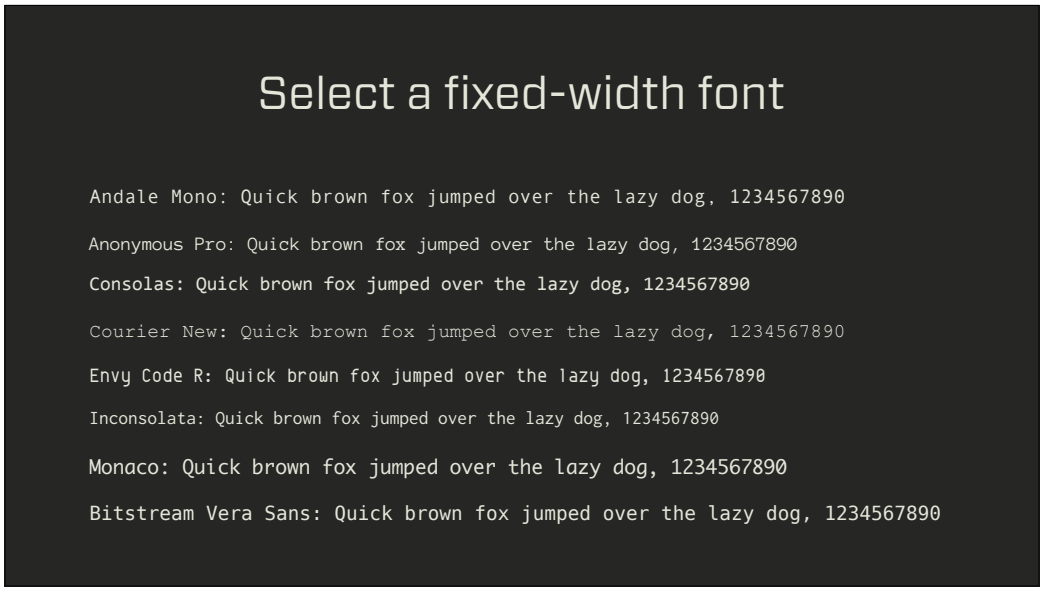
Similarly, we can have clear code and hard to read code.

```
<script>
  var guess, answer, difference;
  answer = 4;
  guess = prompt("What's my number?");
  difference = guess - answer;

  if (guess == answer) {
    console.log("Dagnabit, they got me!");
    alert("You win!");
  }
  else {
    alert("Off by: " + difference);
  }
</script>
```

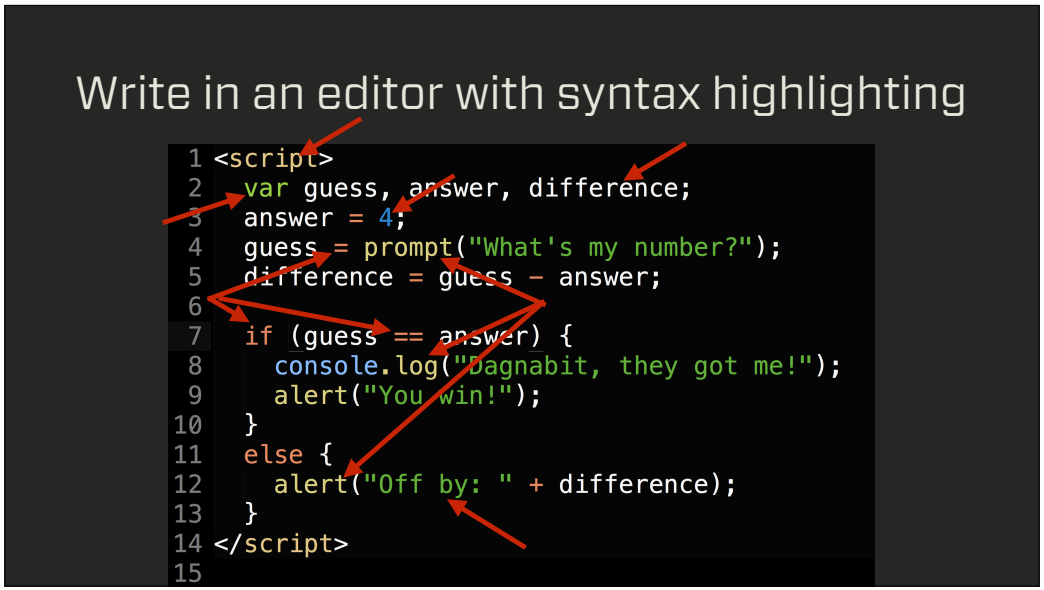
This code does the very same thing. Believe me? I can run it for you. The JavaScript engine doesn't care how it looks.



Write clean code. Here’s the top 5 ways.



Here are some example fonts. I won't belabor this point, and if you are using Sublime, it should default to a fixed width font. Be able to tell your lower-case L from a 1, and a O from a 0 at a glance.

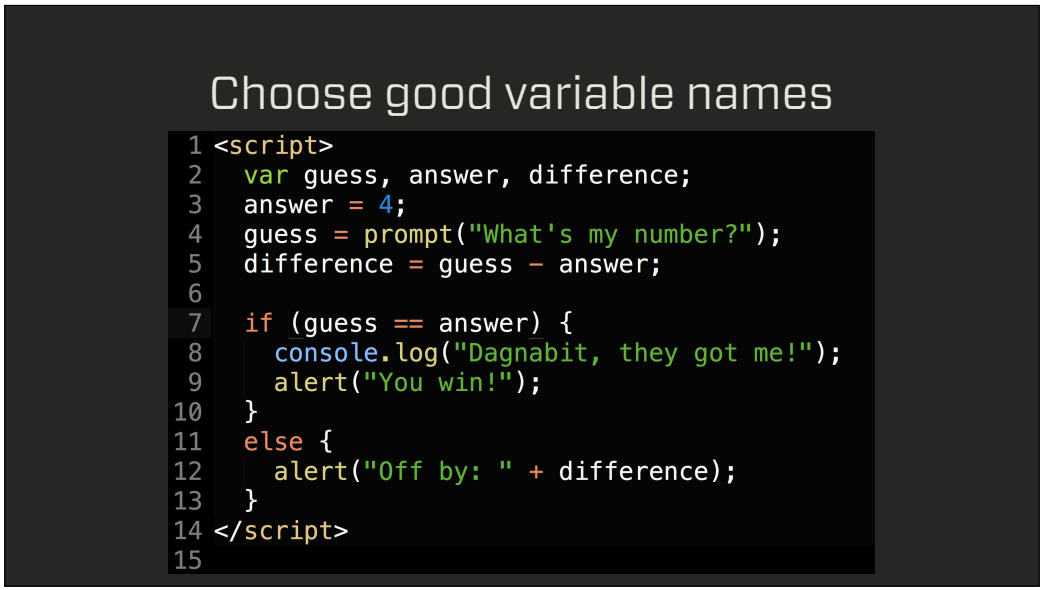


Each piece of this short code snippet is properly colored.

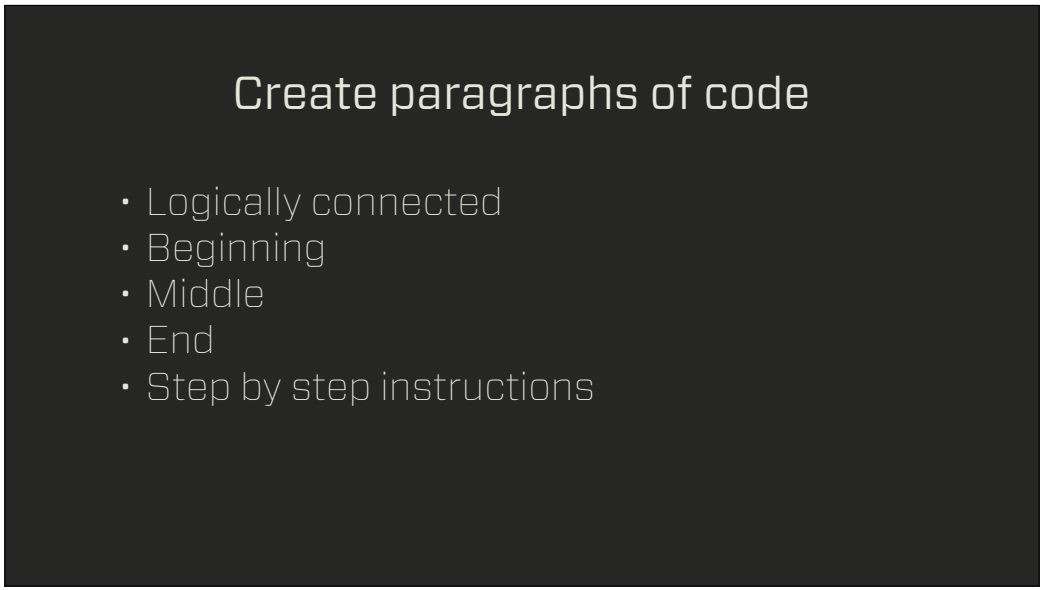
Use whitespace well

```
<script>
  var guess, answer, difference;
  answer = 4;
  guess = prompt("What's my number?");
  difference = guess - answer;

  if (guess == answer) {
    console.log("Dagnabit, they got me!");
    alert("You win!");
  }
  else {
    alert("Off by: " + difference);
  }
</script>
```

Indent 2 spaces for contained code.

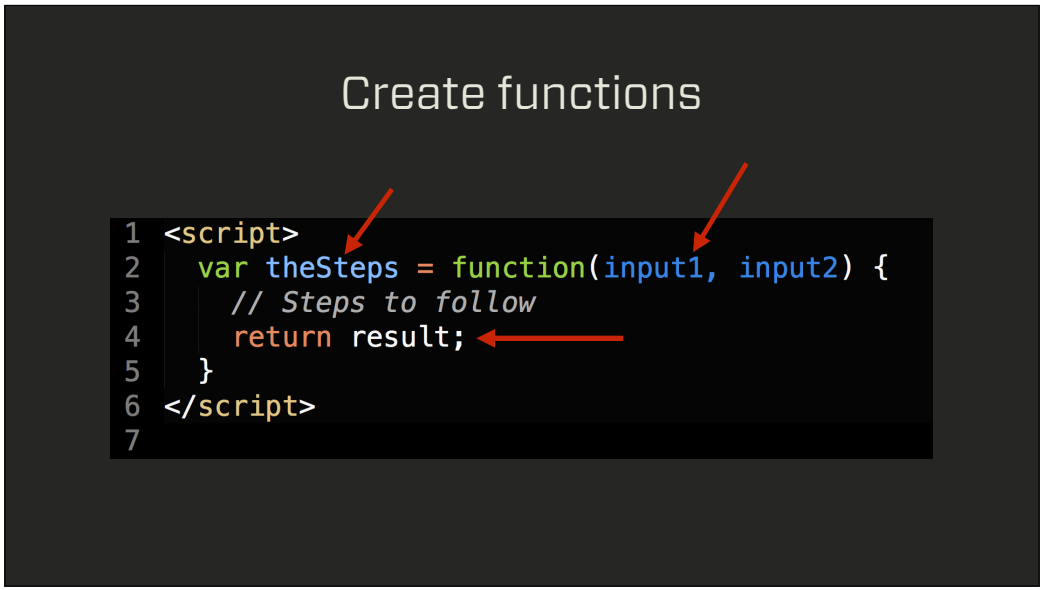


Meaning comes through if we match what we are modeling.



This is what I really wanted to talk about tonight.

What makes a good paragraph? Logically connected ideas, with a beginning, middle and end. Last class was a history lesson, today we are in English class.

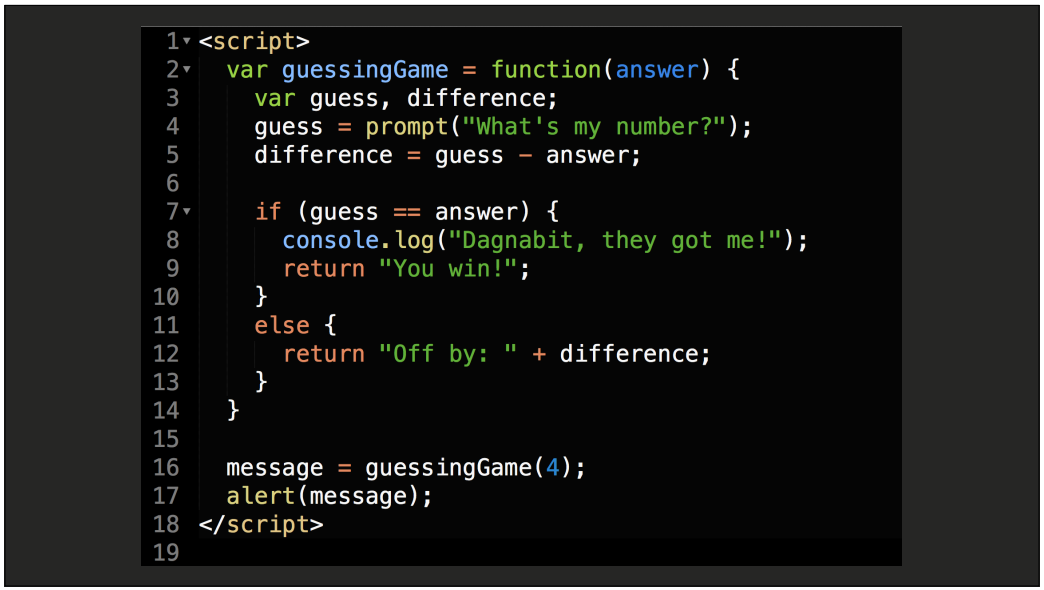


Every function has it’s own signature, a combination of it’s name, arguments, and return value.

Create functions

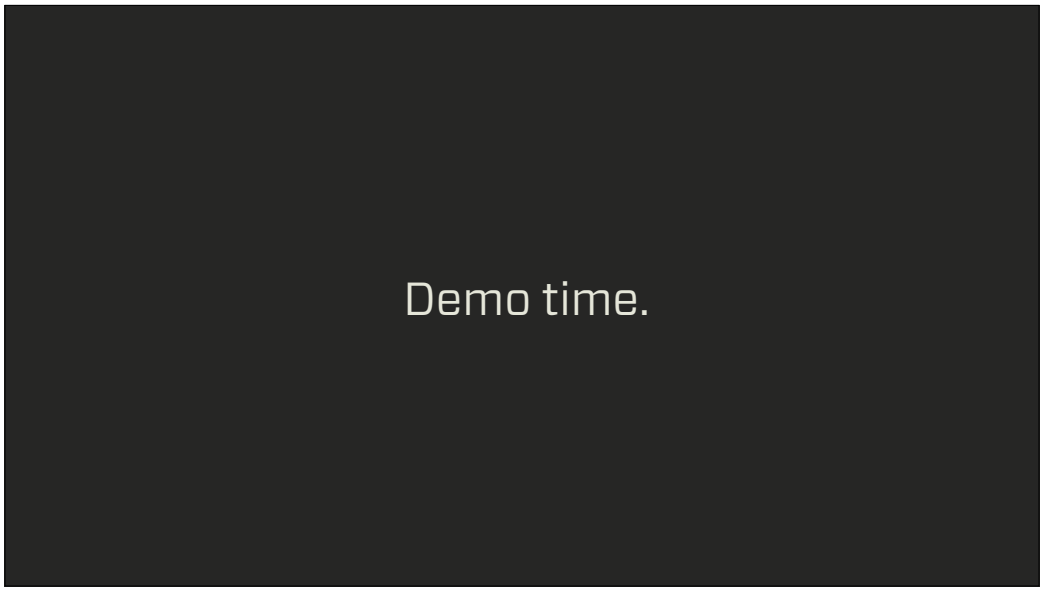
```
<script>
  var findDifference = function(answer, guess) {
    if (answer > guess) {
      return answer - guess;
    }
    else {
      return guess - answer;
    }
  }
</script>
```

We can contain logically connected steps by defining functions.



```
<script>
  var guessingGame = function(answer) {
    var guess, difference;
    guess = prompt("What's my number?");
    difference = guess - answer;

    if (guess == answer) {
      console.log("Dagnabit, they got me!");
      return "You win!";
    }
    else {
      return "Off by: " + difference;
    }
  }

  message = guessingGame(4);
  alert(message);
</script>
```

Our whole game can be wrapped up in a single function. What's the advantage of doing this?



Let's try it! I'll load up some of your code, and see how we can clean it up to make it more readable.

Guess color: https://gist.github.com/brookr/60fac21a4fa7e890b913
Guess GOT: https://gist.github.com/brookr/5163f97b75ff2b088dd9
Guess State: https://github.com/adamrcaldwell/cf-states-guessing/blob/master/topstates.html

Guess Pres: https://gist.github.com/Caldwerl/2915b42192d257c2d8db
[Convert game to function. Add flow control to allow user 3 tries to guess correctly.]