Intro to Coding

Josh Lehman

January 27, 2016

Overview & Logistics

What is Programming?

- Programming *is* the act of writing computer code
- Programming is *about* building and working with ideas

Why JavaScript?

Why not C, C++, Ruby, Perl, Python, Julia, Erlang, Haskell, Clojure, Scheme, Java, PHP, C#, or Swift?

There are lots of great languages, but:

- JS is one of the the most popular languages in the world
- JS runs everywhere!
  - Browsers, servers, microcontrollers, iOS, Android, and more
- JS is *expressive*
  - Can represent ideas in natural/interesting ways

Expectations

- You will be writing actual **code**
- Cannot become a programmer in two hours, but:
  - Get exposure to the basics
  - Can experience the process of coding
  - Have next steps

Class Format

- The lecture is meant to be *interactive*
  - Follow along with examples in lecture
  - Exercises throughout
- **Do not hesitate to ask for help**
  - Public chat in lecture
  - Private chat (see link in video details)

Topics Covered

- **Data**: Primitives & Compound Data
  - Numbers, Strings, Booleans, Arrays, Objects
  - The *what* of our programs
- **Functions**: Building Blocks
  - The *how* of our programs
  - Used to represent *ideas*

Data and Data Structures

"Data" is what our programs store, manipulate, display and compute. Data is the *subject* of our programs.

Question: How can we represent characteristics of a **person** as data (e.g. Facebook profile)?

Primitives: Numbers

```
1 + 3; // we can use operators to perform addition,
7 * 4 / 3 - 17; // multiplication, divisiion and subtraction
Math.floor(7.682); // There are many mathematical functions
```

- Programming is *not* all about math
  - But math is useful
- Order of operations matters (PEMDAS)

Exercises: Numbers

Do the following exercises at the console:

```
// 1. What is the result of the following?
5 + 7 * 8 - 5
// 2. How about this? If they are different, why?
(5 + 7) * (8 - 5)
// 3. Translate your height from inches and feet to centimeters
// (or vice-versa if you know your height in centimeters)
// NOTE: There are 2.54cm per inch.

// e.g. 6 feet, 1 inches is:
((6 * 12) + 1) * 2.54
```

- To open the console:
  - **Mac:** Command + Option + J
  - **Windows/Linux:** Control + Shift + J
- Live 1:1 chat help, click here or #2 on **Important URLS**

Primitives: Strings

```
"Hello, World!"; // a string
"Java" + "Script"; // combine two strings with the "+" operator
"yay strings".length; // the length of the string can be computed
"make me uppercase".toUpperCase(); // Does what you'd expect
```

- Strings are used to represent text
- We use double quotes to represent strings
  - Single quotes work too
- There are lots of useful operations for working with strings

Exercises: Strings

Do the following exercises at the console:

```
// 1. What happens when you use the '+' sign with strings?
"the quick " + "brown fox ..."
// 2. Enter your name as two strings. Use + to combine them (see above)
// 3. Find the length of your name with .length
// 4. Turn your name into upper case.
```

- To open the console:
  - **Mac:** Command + Option + J
  - **Windows/Linux:** Control + Shift + J
- Live 1:1 chat help, click here or #2 on **Important URLS**

Primitives: Booleans

```
true; // this is true
false; // this is false
true && false // logical "and"
true || false // logical "or"
!true // logical "not"
1 > 2;
3 <= 1000000;
4 === 4; // equality
"hello".length >= 5;
```

- Booleans are how we talk about *logic*
- The results of *comparisons* are booleans

Exercises: Booleans

Do the following exercises at the console:

```
// 1. Try the following comparison. What is the result? Any idea why?
"Bob" === "bob"
// 2. How about this one?
4 === "4"
// 3. Try the following comparisons:
1 > 2 || 5 > 2
5 === 5 && 8 > 7
!(1 > 2)
```

- To open the console:
  - **Mac:** Command + Option + J
  - **Windows/Linux:** Control + Shift + J
- Live 1:1 chat help, click here or #2 on **Important URLS**

Variables

```
var ten = 10; // create variable named ‘ten‘ and assign 10 to it.
var myName; // declares a variable without assigning anything
// assign "Josh Lehman" to the existing variable ‘myName‘
myName = "Josh Lehman";
// variable names can be used to reference their values
myName.length > ten;
ten + 5;
ten; // ?
ten = ten + 5; // reassignment
ten; // ?
```

- `var` is used to **create** a new variable
  - The `=` sign is called the *assignment operator*
- Variables are used:
  - To associate *names* with *values*
  - As storage locations
- Variables can be *reassigned*

Exercises: Variables

Do the following exercises at the console:

```
// 1. Change firstName below to contain your first name:
// e.g. var firstName = "Josh"
// 2. Create a variable "lastName" that contains your last name
// 3. Combine firstName and lastName in with '+' and assign it to a
//    variable called fullName
```

- To open the console:

    - **Mac:** Command + Option + J
    - **Windows/Linux:** Control + Shift + J

- Live 1:1 chat help, click here or #2 on **Important URLS**

Interlude: Equality

```
var name = "Josh";
name === "Fred";
name = "Fred";
name === "Fred";
```

The = sign is the *assignment* operator – it *assigns* what's on the right to what's on the right.

We use === to *test for equality.*

Compound Data: Objects

```
var josh = {
  name: { first: "Josh", last: "Lehman" }, // nesting is ok!
  age: 26, // key-value pairs are separated with commas
  gender: "male",
  programmer: true
}
josh.age; // dot notation
josh.name.first;
josh["age"]; // bracket notation
josh["name"]["first"];
josh.name.first = "Joshua"; // we can reassign values!
```

- Used to talk about entities
- Comprised of *key,value* pairs
    - *keys* are usually represented as strings
    - Combine many kinds of data (hence, **compound**)
- Access values by *key*:
    - Dot Notation: `josh.age`, Bracket Notation: `josh["age"]`

Exercises: Objects

These exercises can be found in Codepen here.

- To open the console:
  - **Mac:** Command + Option + J
  - **Windows/Linux:** Control + Shift + J
- Live 1:1 chat help, click here or #2 on **Important URLS**

Compound Data: Arrays

```
var luckyNumbers = [12, 19, 7, 3, 28]; // comma separated!
var animals = ["monkey", "giraffe", "cat", "dog", "platypus"];
var people = [
  {name: "Ben Bitdiddle", age: 27},
  {name: "Eva Lu Ator", age: 32},
  {name: "Alyssa P. Hacker", age: 20},
  {name: "Louis Reasoner", age: 54}
];
animals[0]; // "monkey"
people[2]; // {name: "Alyssa P. Hacker", age: 20}
```

- Arrays are used to represent *many* things
  - The elements of arrays can be *anything*: numbers, strings, objects, etc.
- Usually used to refer to many similar kinds of data
- Elements are *indexed* numerically from 0

All Together Now

```
var josh = {
  name: {
    first: "Josh",
    last: "Lehman"
  },
  age: 26,
  gender: "male",
  programmer: true,
  favoriteTVShows: ["Fargo", "Breaking Bad", "Battlestar Galactica"],
  pets: [{type: "cat", name: "Pal", age: 3, biochipped: true, color: "orange"}]
}
josh.name.last; // "Lehman"
josh.favoriteTVShows[0]; // "Fargo"
josh.pets[0].type; // "cat"
```

- Anything can be accessed!
  - Use combination of **dot** and **bracket** notation

Exercises: All the data!

These exercises can be found in Codepen here.

- To open the console:
  - **Mac:** Command + Option + J
  - **Windows/Linux:** Control + Shift + J
- Live 1:1 chat help, click here or #2 on **Important URLS**

Recap

- **Primitives** are the most basic data types
  - Numbers, Strings, Booleans
- **Variables** are named storage locations
- **Objects** represent data with multiple characteristics
- **Arrays** represent many pieces of data (usually similar)

Basics of Functions

Functions allow us to represent a *task* with a name and parameters.

What are Functions?

```
// "definition" of function named square that accepts one argument
function square(x) {
  // Inside of the curly braces is called the "body"
  return x * x; // return specifies the "result"
}
// "invocation" of square function
square(5); // "invoking" is also known as "calling"
square(square(5));
```

- Functions specify instructions to accomplish some task
- Usually have a *name*
- Usually have *arguments* (the stuff inside the parenthesis)
- Usually `return` a result

Exercises: Basic Functions

These exercises can be found in Codepen here.

- To open the console:
  - **Mac:** Command + Option + J
  - **Windows/Linux:** Control + Shift + J
- Live 1:1 chat help, click here or #2 on **Important URLS**

Functions with Multiple Arguments

```
// Functions frequently have multiple arguments (separated by commas)
function add(a, b) {
  return a + b;
}
add(1, 2); // a: 1, b: 2, 1 + 2 => 3

function divide(x, y) {
  return x / y;
}
// The positions of arguments matters, not the names:
divide(9, 3) // x: 9, y: 3, 9 / 3 => 3
divide(3, 9) // x: 3, y: 9, 3 / 9 => 0.333333333
```

- Functions can have multiple arguments
- **Remember**: arguments are just named placeholders!
  - The values are supplied when the function is *invoked* (or "called")

Exercises: Functions with Multiple Arguments

These exercises can be found in Codepen here.

- To open the console:
  - **Mac:** Command + Option + J
  - **Windows/Linux:** Control + Shift + J
- Live 1:1 chat help, click here or #2 on **Important URLS**

Functions with Data Structures

```
// Functions frequently output data structures...
function makePerson(name, age, cats) {
  var person = {name: name, age: age, cats: cats};
  return person;
}
var johnDoe = makePerson("John Doe", 35, ["Fluffy"]);

// ... will often receive them as arguments...
function aboutPerson(person) {
  return person.name + " is " + person.age +
    " years old, and has " + person.cats.length + " cats.";
}
aboutPerson(johnDoe);

// ... and can manipulate them!
function hadBirthday(person) {
  person.age = person.age + 1;
  return person;
}
hadBirthday(johnDoe);
```

- Data structures and functions work well together

Exercises: Functions with Data Structures

These exercises can be found in Codepen here.

- To open the console:
  - **Mac:** Command + Option + J
  - **Windows/Linux:** Control + Shift + J
- Live 1:1 chat help, click here or #2 on **Important URLS**

Recap

- **Functions** describe how to perform a task given arguments (parameters)
  - Don't solve the same problem multiple times: *use a function*
- Functions can take multiple arguments
- Arguments to and results from functions can be any kind of data