WDI

Introduction to JavaScript

What is JavaScript?

- A client-side scripting language
- Meant to run entirely on the user’s browser
- Defined by the ECMAscript standard, published by the ECMA foundation
- JavaScript != Java. Don’t mess this up or web developers will get very mad at you.

Main uses of JavaScript

- Originally used to handle simple tasks like on-screen calculations

- Can be used to manipulate objects on the page that are in the DOM (Document Object Model), so you can:
  - Show and Hide elements
  - Animate elements
  - Replace elements with other elements
  - Make requests to the server without reloading the page

JavaScript Examples

Let’s walk through some examples of basic JavaScript usage!

(Instructors: See js-examples in WDI repo)

Request flow

or, “where does JavaScript fit in?”

- A user makes a request to the server, server-side scripting languages (Ruby, PHP, Python) are processed
- HTML, JavaScript, and CSS are returned to the user’s browser
- The browser typically runs each line of HTML line-by-line and by extension each JavaScript included in the page in the order indicated
- If using a JavaScript library, always include it before any code that references it!

Writing and running JavaScript

- Use your text editor of choice to write .js files
- Save them and include them in your HTML to run them

```
<script src=”myscript.js”></script>
```

or write JavaScript directly in your HTML:

```
<script>
  alert(‘hi!’);
</script>
```

Comments

- There are two different ways to write comments in JavaScript:

```
// This is a single line comment

/* I’m
a
nifty
multi-line
comment! */
```

Script Output

There are a few different ways to see the output of your script when you run it.

The Console is a built-in tool in your browser where you can run JavaScript code directly. It also allows you to see the output of JavaScript you write in your editor.

Alerts are “pop-ups”. You can see the output in a popup window as soon as the `alert()` function gets called.

Logging to HTML means to change the content of an HTML element with content of your choice, which could be script output.

Why do we need to output?

As you write JavaScript code, you’ll need to periodically check that what your code returns is what you actually want it to return.

Using the Console

- The JavaScript console can be opened in Google Chrome by using Command + Option + J (or Ctrl + Option + J)

- JavaScript can be run line by line inside of the console

- You can also output to the console inside of a . js file using:
```
console.log(“My Content”);
```

Using an Alert

To display an alert, use the `alert();` function. You can do it directly in the console or in a script (a .js file):

```
alert(“Hello World!”);
```

Logging Directly to HTML

You can also change the “innerHTML” of any element, meaning what is contained inside of it:

```
function change(){
  document.getElementById(‘el’).innerHTML = “NEW TEXT”;
}
<div onclick=”change()“ id=”el”>CHANGE ME</div>
```

In the above example, we’re changing the content of a <div> with the ID “el” from “CHANGE ME” to “NEW TEXT”.

See the next slide for a line-by-line breakdown

Code Breakdown

```
// Line 1: Declare a function called change
function change(){
// Line 2: Inside of the function, retrieve the current
// HTML document. Inside of that document, retrieve
// the element with the ID ‘el’ using the getElementById
// function. Set the content inside of the element to say
// “New Text”.
  document.getElementById(‘el’).innerHTML = “NEW TEXT”;
// Line 3: Close the function.
}
// In your HTML document, you’ll need a <div> with the ID
// “el”. The onclick attribute allows you to specify a
// function to run when the element is clicked, in this
// case “change()”
<div onclick=”change()“ id=”el”>CHANGE ME</div>
```

Your first JavaScript

- Create a JavaScript that sends an alert to the user that says “Hello World”

- If you’ve got this down, try inserting “Hello World” into an element on screen instead

Programming Basics

What you’ll need before
you get to the “fun stuff”

Data Types in JavaScript

- Why are we putting any text being logged or alerted inside of quotes?

- By encapsulating our text in quotes, we are telling JavaScript the data inside is a **string**, a basic JavaScript data type

Basic Data Types

- String - `“Hello World”`
- Number - `5, 5.5, 1000` (all numbers in JS are floats)
- Boolean - `true`, `false`
- Undefined (no value)

Variables in JavaScript

- You may have learned about variables in a mathematical context, for instance:
  x = 0
  x + 10 = ?
  Answer: 10

- In JavaScript and in almost any programming language, a variable is simply a container for a value

Variables in JavaScript

- JavaScript variables can be equivalent to any data type

- Variables are defined like so:

```
var name = “Zach”;
var numberOfWidgets = 10;
var isCodingCool = true;
```

Basic Math

- JavaScript can do all basic math. For instance, if you were to run this code in the developer console:

```
10 + 10;
> 20

var x = 100;
x * 40;
> 4000
```

Further Data Types: Arrays

- Arrays are used to hold a collection of data, of any data type. This one is full of strings:

```
["Snoopy", "Charlie Brown", "Patty", "Violet"];
```

- They can hold multiple data types:

```
[11, 15, 25, 48, 79, "elephant"];
```

- Arrays can also be stored in variables:

```
var class_names = ["Julie", "Sophie", "Rob", "John"];
```

Accessing Array Items - Indexes

- Once you’ve declared an array, you may want to retrieve the items inside of it using their indices

- The index of an item inside of an array corresponds to its position from the beginning of the array

- The first item always has an index of 0

- In this array, “charlie brown” has the index of 0 and “snoopy” has the index of 1:

```
["charlie brown", "snoopy"];
```

A R R A Y C E P T I O N

- An array can store other arrays

```
// declare the first array
var toyotas = [“Camry”, “Prius”];
// declare the second array
var porsches = [“Camero”, “Boxer”];
// declare a third array that contains the first
// and second array
var cars = [toyotas, porsches];
```

- This is called a multi-dimensional array

A R R A Y C E P T I O N

To access the items nested inside of a multi-dimensional array, you'll use this syntax:

```
// declare your mult-dimensional array
var cars = [ ["Porsche", "Camero"], ["Camry", "Prius"] ];

// access the first array inside
// then the first item inside that array
console.log(cars[0][0]);

> "Porsche"

// access the first array inside
// then the first item inside that array
console.log(cars[1][0]);

> "Camry"
```

A note on semicolons

- Semicolons are traditionally used to end statements in JavaScript

- Code will still execute without them

- They should be used to indicate the end of a statement

Exercises

- Create a script with two variables assigned to two numbers. Add them together and output the result to the console

- Try to add two strings together and output the result to an alert

- Create a multi-dimensional array related to a subject that interests you. Output two of the items in sub-arrays to the console

Logic

- The control flow of your program

- Think of logic as a river that branches off in a few different ways

- It allows you to make the computer do the thinking for you!

Testing

- Any test returns a boolean `true` or `false`
- To test if two strings are equal:
```
“stringone” === “string two”;
>false
```
- Using three equals signs instead of two also checks the object type
- If you don’t check type, these are both true:
  - `(10-5) == 5;`
  - `(10-5) == “5”;`
- Can cause bugs down the road!

Testing

- To test if two strings are NOT equal:
  ```
  “stringone” !== “string two”;
  > true
  ```

- To test if one number is greater than another:
  ```
  5 > 10;
  > false
  ```

- `<, >, <=, >=` are also valid comparison operators

If...Then...Else...Then

- Now that we have learned testing, we can implement gates into our program

```
if(5>10){
  console.log(“You’ll never see this because 5 is not greater than 10”);
} else{
   console.log(“You will see this because 5 is not greater than 10”);
}
```

Exercise

- Write a program that checks if a variable is less than 10. If it is, alert the user that their variable is less than 10. If it is not, let the user know what the variable was and that it was greater than 10.

- Try running the script and then changing the variable’s value to see how this affects the program output

- If you have extra time, write a similar program to check if a string stored in a variable is the same as another string

If...Then...**Else If...Then...**Else...Then

- Else if is another condition to evaluate in the case where `if` is not true and you have another condition to look at before `else`:

```
if(5>10){
  console.log("You'll never see this because 5 is not
greater than 10");
}else if(5===5){
  console.log("Yes, 5 really is equal to 5.")
}else{
  console.log("You will see this because 5 is not
greater than 10");
}
```

Functions

- A function is a way to encapsulate code for later use
- It can take **arguments**, which are used as variables inside the function

```
function addTwo(some_number){
  return some_number + 2;
}
console.log(addTwo(4));
>6
```

Functions

- Once a function is declared, it can be called later on (invoked) by calling its name and supplying it with any arguments

```
function alertName(somePersonsName){
  return alert(somePersonsName);
}
alertName(“Zach”);
```

Exercises

- Declare a function that takes a name as an argument and tells the user what name they’ve entered, try running it after it has been declared

- Declare a function that takes no arguments but prints something to the console, try running it after it has been declared

- Declare a function that, depending upon which virtual “door” was entered, tells the user they’ve received a different “prize” in an alert. Try running it after it has been declared a few times with each door option.

Identifying Data Types

- Assuming you have a variable with some data stored inside of it, and are unaware of its data type…

- You can ask (or query) the variable for its datatype using the `typeof()` function

```
var yourData = “This is my data.”;
typeof(yourData);
>string
```

Accessing Array Items

- If you’re unsure of an items index number inside of an array, When unsure of an index number

```
 var snoopyPosition = myArr.indexOf(“Snoopy”);
console.log(myArr[snoopyPosition]);
```

```
>”Snoopy”
```

Loops

- A loop is a block of code that gets repeated for a defined amount of iterations (`for` loop) or until a certain condition is met (`while` loop)
- Typically one variable or condition in the loop changes each time it is run

Loops - `for`

```
for(var i = 0; i<10; i++){
  console.log(i)
}

>>0
1
2
3
...
9
```

Loops - `for`

```
beers = [“Lagunitas”, “Peak”]
for(var i = 0; i< beers.length; i++){
  console.log(beers[i])
}

>>”Lagunitas”
“Peak”
```

Exercise - `for` loop

- Create an array filled with two arrays.
- Inside of each of these arrays should be several strings
- Use two loops to display the information (strings) inside of each of these arrays in the console

Loops - **`while`**

```
x = 6
while(x < 10){
   console.log("On number " + x)
   x++;
}
>>6
7
8
9
```

Exercise - `while` loop

Create a while loop that “sings” the classic song “99 Bottles of Root Beer on the Wall”. The code it outputs to the console should look similar to this:

“99 bottles of root beer on the wall, 99 bottles of root beer...take one down, pass it around 98 bottles of root beer on the wall, 98 bottles of root beer on the wall, 98 bottles of root beer...take one down, pass it around 97 bottles of root beer on the wall,” etc., all the way to 0 bottles.