FE101

Getting Comfy with 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

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)
  - Show and Hide elements
  - animate elements
  - replace elements with other elements
  - make requests to the server without reloading the page

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>
  alert('hi!');
</script>
```

or

```
<script src="myscript.js"></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! */
```

Alerts and the Console

- To see the output of your script, you could use an alert
- To call an alert, use:
  `alert("Content of your alert");`
- To bring up the JavaScript console in Chrome, use Command + Option + J
- Arbitrary JavaScript can be run directly in the console line by line
- To log to the JavaScript console, use:
  `console.log("This is logged to the console");`

Basic Data Types

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

Identifying Data Types

- Curious about the type of a piece of data?
- Use the typeof JS function:
  - `yourData = “This is my data.”;`
  - `typeof(yourData);`
    `>string`

Basic Math

- JavaScript can do all basic math:

```
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

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

- They can hold multiple data types

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

- They can be stored in variables

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

Accessing Array Items

- Using an index

```
var myArr = [“Snoopy”, “Charlie Brown”, “Patty”,
“Violet”];
console.log(myArr[0]);
>”Snoopy”
```

- When unsure of an index number

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

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

- An array can store other arrays

```
var toyotas = [“Camry”, “Prius”];
var porsches = [“Camero”, “Boxer”];
var cars = [toyotas, porsches];
console.log(cars[0][0]);>”Camry”
```

- This is called a multi-dimensional array

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 so code can be “minified” later

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");
}
```

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”);
```

Anonymous Functions

- An anonymous function is a function without a name.

```
function(){ alert('hi') }
```

- It's mostly used as an argument to another function

- It can be assigned to a variable

```
var awesome = function(){ alert('hi') }
```

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”
```

Loops - `while`

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

15 Minute Break

JavaScript

A deeper dive

Variable Scope

Some examples from http://stackoverflow.com/questions/500431/javascript-variable-scope

- In order to program JavaScript well, you need to understand the scope of JavaScript variables

- As it relates to JavaScript, scope generally refers to the value of a variable in a specific piece of your code

Global Scope

- A variable that can be accessed globally, within almost any part of your script

```
var z = 1;

function print_it(){
  console.log(z); //Will return 1
}
```

Local Scope

A variable that only pertains to the function you are currently in

```
var z = 3;

function some_f(z){
  console.log(z);
}
some_f(10)
>>10
```

Local Scope

A variable that only pertains to the function you are currently in

```
var z = 3;

function some_f(){
  z = 20
  console.log(z);
}
some_f()
>>20
```

Scope Exercise

Create a script that exemplifies global scope and local scope by logging a variable to the console.

No block scope in JS

There is no such thing as block scope in JavaScript.

```
var c = 10;
function hallo(){
  if(true){
    var c = 2;
  }
  console.log(c);
  //returns 2, not the global value 10
  //implication: variables don't have a
   separate scope within a logical block (in
   bold)
}
```

Function Scope

An argument is only accessible within the function it gets declared in

```
function haha(argument_uno){
  console.log(argument_uno);
}

haha(“hello”);
>>”hello”
console.log(argument_uno);
>>ReferenceError: argument_uno is not
defined
```

Exercises

- Write the following JavaScript functions:
  - Check if a string is a palindrome
  - Accepts a string as an argument and checks it for vowels
  - When your name is entered in a textbox and the user presses a button, an alert is displayed.

More Exercises

For the adventurous

- Write a program to validate a date input. Given a date (06/15/1990, for instance), make sure that it's in mm/dd/yyyy format.

- Write a program to find the most frequent item in an array