RENO COLLECTIVE

FULL SPEED

N V

AHEAD

ACADEMY

Introduction to Javascript

Statements

```
console.log(‘Hello World!’);
```

Javascript Console

```
console.log('Hello World!');
```

```
> Hello World
```

For now we can view the console in codepen.io or repl.it

Variables

Declare a variable **x**, then initialize it with a value of **5**.

```
var x;
x = 5;
console.log(x);
```

Declaring and initializing on the same line.

```
var y = 5;
console.log(y);

y = 6;
console.log(y)
```

Primitive Data Types

• **string**	`var hello = ‘Hello World’;`
• **number**	`var myAge = 28;`
• **boolean**	`var lightOn = false;`
• **undefined**	`var vacation;`
• **null**	`var vacation = null;`

Naming Variables

- Begin with letters, $ or _
- Never start with a number
- Only contain letters, numbers, $ and _
- Case sensitive
- Avoid using keywords or reserved words
- Choose names that provide meaning
- Use camelCase instead of _
- Be consistent

Expressions

```
var sum = 2 + 3;

var product = 3 * 2;

var name = ‘Colin’;

var greeting = ‘Hello ’ + name;
```

Variable Types

A variable can only be of one type but
Javascript detects the type based on the value.

```
var x;

x = 2;

console.log(typeof x);   // number

x = ‘hello world’;

console.log(typeof x);   // string
```

Code Comments

```
// Single-line comments using two forward slashes

var x = 2 + 2;

/*
Multi-line comments using a slash and a star. Ended
with a star and a slash.
*/

var y = “Hello World”;
```

Functions

Functions are blocks of code that are defined to perform a specific task. You can then call when needed.

```
function outputName() {
   console.log(‘Hello Colin’);
}

outputName();
```

```
> Hello Colin
```

Function Arguments

```
function outputName(name) {
    console.log(‘Hello ’, name);
}

outputName(‘Chris’);
```

```
> Hello Chris
```

```
var player1 = ‘Josh’;
```

```
> Hello Josh
```

Function Arguments

Functions can be defined with any number of arguments.

```
function printSum(num1, num2) {
    console.log(num1 + num2);
}

printSum(2, 4);
```

```
> 6
```

Return Values

Functions can return a value to exit the function and provide a value to the code that called the function.

```
function sum(num1, num2) {
   return num1 + num2;
}

var result = sum(2, 4);
console.log(result);
```

```
> 6
```

Mix & Match

You can combine most of what we’ve learned so far to call functions inside of expressions or call functions within other functions.

```
function sum(num1, num2) {
   return num1 + num2;
}

var sums = sum(4,5) + sum(5,6);

var result = sum(sum(4,5), sum(5,6));
```

Variable Scope

Variables in Javascript have what is called “function” scope. If you define a variable inside of a function, it is only available inside of that function.

Local Variables

```
function sum(num1, num2) {
    var localSum = num1 + num2;
    console.log('The sum is: ' + localSum);
    return num1 + num2;
}

sum(4,5);
console.log(localSum);
```

```
> The sum is: 9
ReferenceError: localSum is not defined
```

Global Variables

```
var globalSum;
function sum(num1, num2) {
    var globalSum = num1 + num2;
    console.log('The sum is: ' + globalSum);
    return globalSum;
}

sum(4,5);
console.log(globalSum);
```

```
> The sum is: 9
9
```

Scope Precedence

```
var g = "global";

function run() {
  var l = "local";
  var g = "in here!";
  console.log(g + " inside go");
}

run();
console.log(g + " outside go");
```

Control Flow

```
if (expression) {
    // code block executes if expression is true
}
```

Control Flow

```
var x = 10;

if (x < 5) {
    console.log("x is less than 10");
}
```

Control Flow

```
var x = 10;

if (x < 5) {
    console.log(“x is less than 10”);
} else {
    console.log(“x is greater than or equal to 10”);
}
```