Week 3 Workshop

Web Development Fundamentals
HTML, CSS, and JavaScript

Agenda

Activity	Time	~Start
Get Prepared: Log in to Nucamp Learning Portal • Slack • Screenshare	10 minutes	9:00am
Check-In	10 minutes	9:10am
Review	60 minutes	9:20am
Task 1	40 minutes	10:20am
BREAK	15 minutes	11:00am
Task 2 & 3	90 minutes	11:15am
Check-Out	15 minutes	12:45pm

Check-In

- How was this week for you? Any particular challenges or accomplishments?
- Did you understand the Exercises and were you able to complete them?
- How were the Challenges and Quiz this week?
- We know that this was a difficult week for many. Please ask if you have questions.

Week 3 Review - Overview

- What is JavaScript?
- HTML: The script element
- HTML: The onclick attribute
- Variables
- Data types
- Functions
- Function Parameters & Arguments
- The JavaScript console
- If … Else If … Else
- Comparison Operators Logical Operators
- Truthy & Falsy
- Switch Statements
- += -= ++ --
- While Loops
- Arrays
- Scope
- Math.random() & Math.floor()

What is JavaScript?

- Programming language originally created to run inside browsers
- Most popular programming language for web development, used in browsers, desktops, servers, mobile apps
- ECMAScript: Official specification for JavaScript – last major version was in 2015, called ECMAScript 2015 or ES6
  - Current version is ES9, but changes since ES6 have been minor
- Many technologies use JavaScript (jQuery, Node.js, React, etc); you must first learn "vanilla JavaScript" to use them

HTML: The script element

- Use <script> element to add JavaScript to HTML page
- Add JavaScript between <script></script> tags
- ...or link to external JS file: <script src="index.js"></script>
- You can link to multiple JS files
- Generally, JavaScript inside <script> tags or in external JS file that is *not* inside a function will run automatically when the page is loaded

HTML: The onclick attribute

- Add onclick attribute to HTML element such as button to run JS function when element is clicked:

```
<button type="button" onclick="runFunction()">Click Me</button>
```

- There are multiple ways to trigger JavaScript from an HTML page, this is one way

Variables

- A named container for some value
- Create/Declare a variable using let or const
  - let for variables that will have their values reassigned
  - const for variables whose values will be assigned only once
- Pre-ES6 variable declaration keyword var is commonly seen in older codebases – avoid when writing new code
- Values stored in variables can be of several different data types

Variables (cont)

- Use the assignment operator = to set a variable's value

- You learned about: number, string, boolean

```
const myNum = 1;           // number
const myString = 'foo';    // string with single quotes
const myString2 = "5";     // string with double quotes
const myBoolean = true;    // boolean
const myBoolean = false;   // boolean

// Combine / Concatenate Strings into a single value with the "+" operator
const myStringCombined = 'foo' + "bar";  // result "foobar"
```

"5" is a String and not a number since it is surrounded by quotes

boolean can be either **true** or **false**

Use the + operator to combine (concatenate) strings

- Two more data types: null and undefined
  - null is an intended non-value set by the programmer
  - undefined is the value of a variable that has been declared but not initialized

- There are other data types that will not be covered for this introductory class

Variables (cont)

- The first time you assign a value to a variable is called initialization

- You must initialize const variables at declaration:

```
const num = 1;   // CORRECT

const num2;      // BAD
```

Declaration: Defining a name for your variable for use in your code (var, **let** & **const**). i.e. **let varName;**

Assignment: Sets/re-sets the value of your variable(**=** , +=, -=, *=, and /=). i.e. **varName = "hello"**

Initialization: Specifying an initial value for your variable to start with. i.e. **let varName = "hello";**

- You *can* declare let variables then initialize later, but best practice: initialize at declaration to prevent issues with undefined variables

```
let x;
let x = 0;
```
<-- OK, technically permitted

<-- Better

- Only use **let** keyword when you first create the variable

- To assign/reassign its value later after declaration, use only the variable name

```
x = 2;
```

Functions

- A segment of code that can be grouped together, given a name, and called by that name from other places in the code

- Multiple ways to define a function, simplest way is with function declaration syntax:

```
function sayHello(name) {
  console.log('Hello ' + name);
}
```

- Defining/declaring a function does not run it. It must be called.

- Call a function (run the code inside it) with the function's name followed by an argument list:

```
sayHello('John');
```

Functions – Parameters and Arguments

- Function definitions must include a parameter list
  - Variable names for values that will be passed in when function is called
- Function call's pass an argument list for the parameter values

Pass arguments **TO** functions when calling it

The arguments passed then become parameters (variables) to be used inside a function

- Parameter list can be empty:

```
function myFn1() {...}  // ... just means some code here
```

- If so, function is called with empty argument list:

```
myFn1();  // No arguments passed since myFn1() has an empty parameter list
```

- Otherwise, call with arguments that correspond to parameters:

```
// greetings function accepts/expects 2 arguments to be passed to it
// firstName and lastName are Parameters that will be local variables in the function
function greetings(firstName, lastName) {
  console.log("Grettings " + firstName + " " + lastName);
}
```

```
// Call "greetings" function and pass "John" as 1st argument and "Doe" as 2nd argument
greetings("John", "Doe");
```

No need to declare these variables using let or const! The parameter list does it for you

Scope

- Variables declared with let and const are block scoped

  Only exist inside the code block { … } in which they were declared, such as a function, if, switch, or while/do…while block.

- Variables declared with var are function scoped

  Function scope is like block scope, but only for functions

- Child blocks inherit their parent blocks' variables

- Variables declared outside of any code blocks are global and can be accessed from anywhere – use sparingly

```
// Global variable (no blocks { })
let globalVariable = "Hello";


function greeting(fname, lname){
  /* Anything in between { } here is block scope
  (children can inherit) */
  let fullName = fname + " " + lname;

  if (fullName){
    /* Access parent global variable from here as
       well as function variable (fullName) */
    console.log(globalVariable + " " + fullName);
    var fnAccess = "accessible by parent fn bad practice";
    /* if you change var to let then you cannot access
       this which is considered best practice and safer */
  }

  // access the "var" variable inside the "if{...}" block
  console.log(fnAccess);

}

greeting("John", "Doe"); // Call the greetings functions
```

The JavaScript console

- Three primary uses:
  1. View error/warning messages
  2. Log your own messages using console.log('…');
  3. Test out small pieces of JavaScript and have their values immediately evaluated and echoed back to you

```
> console.log("Hello World");
  Hello World
```

If … Else If … Else

- Conditional statement, allows forks in your code

```
if (condition) {
    // Code to execute if condition evaluates as true …
} else if (condition2) {
    // Code to execute if condition2 evaluates as true …
} else {
    // Code to execute if neither condition1 nor condition2 were true …
}
```

- else if and else are optional, you do not need them
  - You can have either or both, following an if block
  - You can have multiple else if blocks
  - You can have only one else block at the very end

Comparison Operators

- Equality Operators
  - **Strict equality** (aka triple equals/identity): ===
  - Loose equality (aka double equals/equality): ==
  - **Strict inequality** (aka non-identity): !==
  - Loose inequality (aka inequality): !=

Discuss: What's the difference between the strict and loose versions of the equality operators, and which are best practice to use?

== does NOT evaluate the data type (i.e 1 == "1" will return true)

=== is a strict equality where the data types must match (i.e 1 === "1" will return false)

=== Strict is best practice

- Relational Operators
  - > >= < <=
  - Greater than, greater than or equal to, less than, less than or equal to
  - Works as you would expect with numbers
  - Works in lexicographical order with strings; 'a' is lower/less than 'z'

Truthy & Falsy

- Boolean values **true** and **false** are of the Boolean data type only
- The concept of **truthy** and **falsy** mean that if a value was converted to the Boolean data type, it would be **true** or **false**.
- Example: the number 3 is truthy, the number 0 is falsy

Discuss: Is the number -1 falsy?

No, -1 is truthy

Falsy – Any value that is false, 0, an empty string (""), undefined, null, and NaN will be interpreted as false

Truthy – Any other value that NOT Falsy will be interpreted as true

Truthy & Falsy (cont)

- There are only 6 **falsy** values:
  - false
  - null
  - undefined
  - empty string: "" and ''
  - 0
  - NaN (Not a Number)

- Everything else is **truthy**!

Logical Operators

- Logical AND &&: Returns first falsy value or last truthy value

Discuss: What is returned from evaluating **(true && (3 >= 5))**?

false

(For practice, enter into your JavaScript console to confirm your answer)

- Logical OR ||: Returns first truthy value or last falsy value

Discuss: What is returned from evaluating **(false || (5 - 10))**?

-5

-5 is the first truthy value and is returned

Logical Operators

- Logical Not !: Coerces its operand to Boolean then returns its opposite

Discuss: What is returned from evaluating

- !(true && false)? True - !false (not false) evaluates to true
- !true && false? False - false && false = false

- Double Not !!

Discuss: What is this used as a shorthand for and why/how does it work?

Can coerce (implicit type conversion) a value to its truthy/falsy Boolean value

The first "!" will convert it to the oppisite boolean and then the second "!" converts it back to it's actual truthy/falsy boolean value

Switch

- Conditional statement – evaluates an expression depending on its value, then executes one of multiple case clauses and an optional default clause:

```
switch(myNum) {
  case 1: console.log('In case 1');
    break;
  case 2: console.log('In case 2');
    break;
  case 3: console.log('In case 3');
    break;
  default: console.log('In default');
}
```

```
switch(myString) {
  case 'coffee': console.log('Contains caffeine');
    break;
  case 'black tea': console.log('Contains caffeine');
    break;
  case 'lemonade': console.log('No caffeine');
    break;
  default: console.log('Drink not recognized');
    break;
}
```

- Once the program enters a case, it will executing all following statements until it reaches the end of the switch block, or a break, *even the statements for other cases.*
- Always use a break unless you know what you're doing and you want that behavior.
- default clause is like the "else" in an if statement, will run if nothing else matches, best practice is to always use it

More Operators: += -= ++ --

- `+=` and `-=` are binary operators

```
let x = 3;
x += 5;
//x is now 8
x -= 2;
//x is now 6
```

Same as x = x + 5

Same as x = x - 2

- `++` and `--` are unary operators that only add or subtract 1
  - can be used prefix (++variable) or postfix (variable++) and have different behaviors
    - ++varName will increase "varName" by +1 first and then perform an evaluation
    - varName++ will evaluate "varName" at it's current value first and then increase by +1 after that

```
let varName = 1;
console.log(varName++); // 1
console.log(varName); // 2
```

```
let varName = 1;
console.log(++varName); // 2
console.log(varName); // 2
```

  - recommended to use **+= 1** and **-= 1** instead of these in most cases, more clear

While Loops

- Repeat a block of code until a condition evaluates as false

```
let i = 0;
while (i < 5) {
    i += 1;
    console.log('i is', i);
}
```

```
let i = 0;
while (i < 5){
  i++;  // increment
  console.log('i is', i);
}
```

```
let i = 0;
while (i < 5){
  ++i;  // increment
  console.log('i is', i);
}
```

```
i is 1
i is 2
i is 3
i is 4
i is 5
```

Results for all 3

```
let i = 0;
while (i < 5){
  console.log('i is', i++); //increment after statement
}
```

```
i is 0
i is 1
i is 2
i is 3
i is 4
```

Prints i then increment
0 is included and 5 is excluded

Do … While Loops

- Variant of while loops where the code block always executes at least once, even if the while condition is false

```
let i = 0;
while(i) { // i is falsy, so loop will not be entered
    console.log('Got in the loop');
}
undefined
```

```
let i = 0;
do {
    console.log('Got in the loop');
} while(i); // do ... while, so loop will be entered once
Got in the loop
```

Arrays

- Numerically indexed list of values: **[item1, item2, item3, …]**
- Zero-indexed – **index starts at 0, not 1**
- **Example:**

```
const fruits = ['apple', 'banana', 'cherry']
```

  - 'apple' is at index 0 and can be accessed with fruits[0]
  - 'banana' is at index 1 and can be accessed with fruits[1]
  - 'cherry' is at index 2 and can be accessed with fruits[2]
- *arrayname*.length is an **array property** will give you the count of items in the array
- For example: fruits.length is 3
- You can modify the value: fruits[1] = 'boysenberry'; will result in the array being changed to: ['apple', 'boysenberry', 'cherry']

Array Methods

- Some are mutator methods – they change the array
- Others are not – only access the array
- Some have parameters, others don't
- Most will return some value, different for each method
- Very useful – there are many, it will take time to learn them all

Array Methods – push(), pop(), unshift(), shift()

- push() adds an item to end of array, returns the new array length
  - Use with argument of item(s) to add

```
arr1.push("newItem1");
```

- pop() removes an item from end of array, returns the removed item
  - No arguments

```
arr1.pop();  // removes/grabs last item in arr1
```

- unshift() adds 1 or more item to start of array, returns new array length
  - Use with argument of item(s) to add

```
arr1.unshift("newItem1");
```

- shift() removes an item from start of array, returns removed item
  - No arguments

```
arr1.shift();  // removes/grabs first item in arr1
```

- All four of these are **mutator** methods

Discuss: Which two of these four affect the index of all other items in the array and why?

unshift() - When adding to the start of an array your increasing the other item indexes by the number of items being added
shift() - When removing from the start of an array your decreasing the other item indexes by 1

Array Methods – join()

- **join()** – returns a string with the array items
  - Takes an argument of a string that will be used as the separator between array items in the returned string
  - If no argument is given, comma is used
  - Does not mutate the original array – the array fruits will still be the same after you use join() on it

```
const fruits = ['apple', 'banana', 'cherry'];
console.log(fruits.join()); // return with commas
```

default → `apple,banana,cherry`

```
const fruits = ['apple', 'banana', 'cherry'];
console.log(fruits.join(" ")); // return with a space seperator
```

→ `apple banana cherry`

```
const fruits = ['apple', 'banana', 'cherry'];
console.log(fruits.join("|")); // return with a pipe "|" seperator
```

→ `apple|banana|cherry`

Array Methods: includes(), indexOf()

- Both array methods will check to see if a value exists in an array
- includes(*value to check for*) will return true if so, false if not
- indexOf(*value to check for*) will return the numeric index of the item if it exists in the array, and –1 if not
- Example: for an array of:

```
fruits = ['apple', 'banana', 'cherry'];
```

  - fruits.includes('banana') would return true
  - fruits.indexOf('banana') would return 1

Discuss: Why does **indexOf** return -1 and not 0 for a not found item?

0 is a valid index number – it's the first item in the array

Math.random()

- **Math.random()** generates a random number between 0 and 1 such as:
  - 0.03439834432
  - 0.99999999999
  - 0
- Potential values **include 0** but **not 1**
- If you want a value between 0 and a max number (not inclusive of the max number), multiply by the max number:
  - **Math.random() * 10** would generate a random number between 0 and 9.9999999999...

Math.floor()

- **Math.floor()** takes a number as an argument and returns an integer
  - **Math.floor(9.9999)** would return 9
  - **Math.floor(9.1111)** would return 9
  - **Math.floor(3.14)** would return 3

- Use it along with **Math.random()** to generate a random integer:
  - **Math.floor(Math.random() * 10)** would generate a random integer between 0 and 9, including 0 and 9

```
Math.floor(Math.random() * 10);
```

- Add 1 to the result to get a value that's between 1 and the max number, inclusive of the max number:
  - **Math.floor(Math.random() * 10) + 1** would generate a random integer between 1 and 10, including 1 and 10

```
Math.floor(Math.random() * 10) + 1;
```

This Week's Tasks

- If we have extra time before the Workshop then feel free to bring up any unresolved questions, and to discuss any Challenge Questions or Code Challenges.

- Otherwise, please start the Workshop Assignment and save the discussion for after the assignment is finished, or online.

Workshop Assignment

- It's time to start the workshop assignment!

- Break out into groups of 2-3.
  - Sit near your workshop partner(s) in in person
  - For online Workshops your instructor may break you out into different virtual rooms

- Work closely with each other.
  - Don't forget that the 20-minute rule becomes the 10-minute rule during workshops!
  - 10-minute rule does *not* apply to talking to your partner(s). Work together throughout. This will be useful practice for working with teams in real life.

- Follow the workshop instructions very closely.
  - Talk to your instructor if any of the instructions are unclear to you.

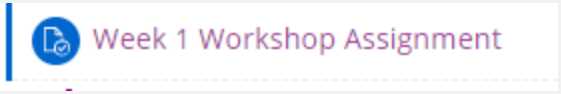
Assignment Submission & Check-Out

- Submit the **color-guessing-game.html** page at the bottom of the assignment page in the learning portal.

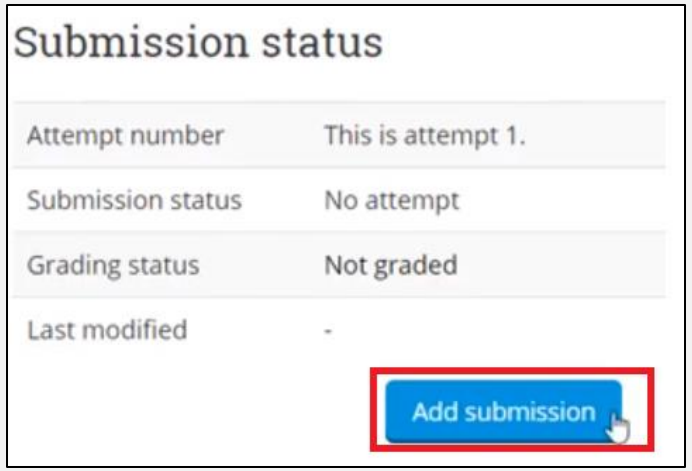
- Example instruction on the next slide

Submitting Your Assignment

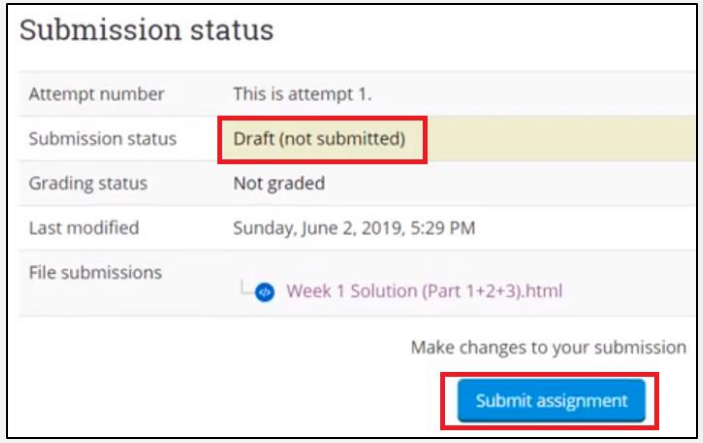
- Go to https://learn.nucamp.co
  - Click "**Workshop Assignment: Students' Work**"



  - Upload your work by clicking "**Add Submission**", select the file, and then click "save"



- Note that your work is in Draft status
  - Click "**Submit assignment**" to submit it



Happy learning!