

CLASS FORMAT

LECTURE

Review and Function Basics

EXERCISE

Code Refactor

GROUP ACTIVITY

Soliciting Feedback

LED BY

Eric Hanson

LECTURE

Advanced ES6 Features

QUICK REVIEW

TEMPLATE LITERALS

- Backticks can be used to surround strings instead of single or double quotes.
- Can span multiple lines.
- Insert variables or expressions with: \${ }

ARRAY SYNTAX

- Array elements are indexed meaning they are assigned a number starting with 0.
- Individual elements are accessed with the name of the variable followed by the number of the element inside square brackets.

OBJECT SYNTAX

```
const person = {
  firstName: "Jane", /* property: value */
 lastName: "Smith",
  age: 30,
  eyes: "blue",
 hair: "brown" /* no comma after the last property */
};
person.firstName /* dot notation */
person['firstName'] /* square bracket notation */
```

- Objects are surrounded by curly braces.
- Objects store data in key/value pairs.
- Each object property is separated by a comma.

FOR-OF LOOP SYNTAX

```
const students = [
    {firstName: 'Karli', lastName: 'Davis'},
    {firstName: 'Christian', lastName: 'Martucci'},
    {firstName: 'Leona', lastName: 'Harrelle'},
    {firstName: 'Gerry', lastName: 'Connor'}
];
let honorList = '';
        2
for (let student of students) {
  honorList += `
    ${student.lastName}, ${student.firstName}
                5
$('#honor-roll').append(honorList);
```

- 1. The **for** keyword
- Declare a variable to hold the current value on each loop
- 3. The of keyword
- 4. The array to iterate
- 5. Current element values

FUNCTION BASICS OBJECTIVES

- Understand why to use functions
- Learn how to create and use functions
- Understand what function hoisting is

FUNCTION BASICS

WHAT ARE FUNCTIONS

A function is a block of code within our overall script that performs some task. We use functions to make our code **DRY**. DRY code is more readable, reuseable and maintainable.



DRY is a popular acronym in programming that stands for Don't Repeat Yourself. The opposite of DRY is WET code (Write Everything Twice).

YOU ALREADY KNOW FUNCTIONS

We've been using a type of function called an **anonymous function** inside of our event listeners.

```
$('button').click(function(){
    /* This function wraps the code to execute */
});
```

FUNCTION TYPES

- Anonymous Functions: Anonymous functions are most often run when triggered by a specific event or as a callback.
- Named Functions: Named functions are executed when called by name.
- IIFE: Immediately Invoked Function Expressions are run the moment the Javascript engine encounters them.

FUNCTION DECLARATION SYNTAX

```
function functionName(arg1, arg2) {
   /* Code block of stuff to do when this function is called. */
}
```

- Start with the function keyword
- Named functions are given a name that follows the function keyword
- The function keyword or name is followed by (), which may or may not contain any arguments.
- The entire code block is then wrapped in {}.

SAY HELLO!

```
function sayHello() {
  console.log('Hello!');
}
console.log(sayHello); /* outputs the function code */
sayHello(); /* outputs "Hello! */
```

- When a functions or method is run, we say they are called, executed or invoked
- Named functions are called with the name followed by () wherever we want the function to execute in our overall script.

ADDING AN ARGUMENT

```
function sayHello(name) {
  console.log(`Hello ${name}!`);
}
sayHello('Jen'); /* outputs "Hello Jen!" */
sayHello('Kelly'); /* outputs "Hello Kelly!" */
```

- Arguments allow us to get data into our functions
- The argument name acts as a variable inside the function and is replaced with whatever data we give the function when called.

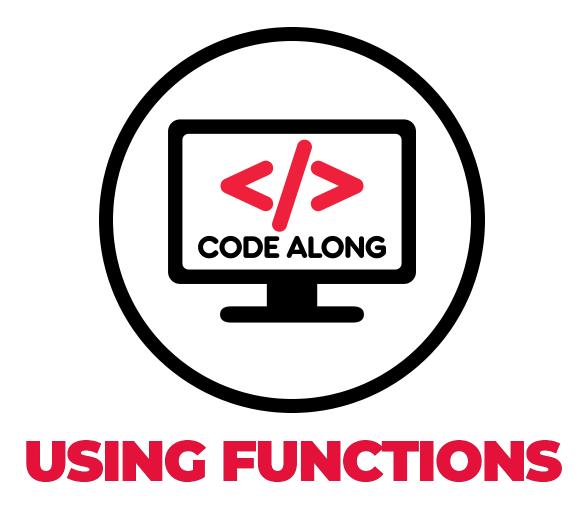
MULTIPLE ARGUMENTS

```
function sayHello(firstName, lastName) {
  console.log(`Hello ${firstName} ${lastName}!`);
}
sayHello('Jen', 'Meade'); /* outputs "Hello Jen Meade!" */
```

- Multiple arguments are separated with commas
- When we have multiple arguments, the order of the data we pass into the function matters!

ARGUMENTS SCOPE

```
let fname = 'Jen';
let lname = 'Meade';
function sayHello(fname, lname) {
  console.log('Hello, ' + fname +' '+ lname + '!');
sayHello(); /* output: "Hello, undefined undefined!" */
sayHello(fname, lname); /* output: "Hello, Jen Meade!" */
sayHello('James', 'Bond'); /* output: "Hello, James Bond!" */
console.log(fname, lname); /* ouput: "Jen" "Meade" */
```



FUNCTIONS THAT GIVE BACK

```
let lowerJen = convertText('Jen', 'lowercase');
function convertText(string, type) {
  if (type === 'uppercase') {
    return string.toUpperCase();
  } else if (type === 'lowercase') {
    return string.toLowerCase();
    console.log('ran lowercase operation'); /* never executes! */
console.log(lowerJen); /* output: jen */
```

- Functions can be made to return a value to the caller using the return keyword.
- The return statement **exits** the function without running any code that follows it within that function.

REFACTOR LAB

//codepen.io/jme11/embed/GwYYGr/?height=265&theme-id=default&default-tab=html,result&embed-version=2&editable=true

https://codepen.io/jmell/pen/GwYYGr

GETWINNER SOLUTION

```
function getWinner(humanPlay, computerPlay) {
 let wonGame = (humanPlay === 'rock' && computerPlay === 'scissors') | |
      (humanPlay === 'scissors' && computerPlay === 'paper') ||
      (humanPlay === 'paper' && computerPlay === 'rock');
 if (humanPlay === computerPlay) {
   updateScreen(computerPlay, "You tied :- | ");
 } else if (wonGame) {
   ++humanScore;
   updateScreen(computerPlay, "You won! :-)");
 } else {
   ++computerScore;
   updateScreen(computerPlay, "You lost :-(");
```

UPDATESCREEN SOLUTION

```
function updateScreen(computerPlay, result) {
    $computerPlay.text(`The bot played ${computerPlay}`);
    $results.text(result);
    $humanScore.text(humanScore);
    $computerScore.text(computerScore);
}
```

GROUP ACTIVITY

ES6 FEATURES

HOISTING

- Variables declarations made with var hoist to the top of their scope
- Function definitions hoist the function to the top of the scope
- Function expressions follow the variable rules
- ES6 fixes this because let and const don't hoist

BLOCK SCOPE

- Both let and const have block scope
- Variables defined with var have functional scope only

DEFAULT PARAMETERS

```
function multiply(a, b = 1) {
  return a * b;
}

multiply(6); /* output: 6 */
multiply(8,2); /* output: 16 */
```

DESTRUCTURING

```
function firstItem([first, second] = ['luke', 'skywalker']) {
   return first;
}

/* Another example in for of */

const people = ['Jen', 'John', 'Terrell', 'Kevin'];

for (let [index, value] of people.entries()) {
   console.log(index, value);
}
```

FAT ARROW SYNTAX

```
arr = [2,4,6,8,10];
let newArr = arr.map(function(element) {
   return element + 2;
});
let newArr2 = arr.map(element => element + 2);
```

- Drop the function keyword
- Use => following arguments
- Drop the parens if there's only one argument
- Drop the curly braces if there's only one line
- Drop the return if there's only one line



```
let agent = {
 firstName: "James",
 lastName: "Bond",
  preferences: ["things that blow up", "Aston Martins", "tuxedos"],
 printPreferences() {
    this.preferences.forEach(function(pref){
      console.log( this.firstName +" prefers "+ pref );
    }).bind(this); // this falls out of scope
let agent1 = {
 firstName: "James",
 lastName: "Bond",
  preferences: ["things that blow up", "Aston Martins", "tuxedos"],
 printPreferences() {
    this.preferences.forEach(pref => {
      console.log( this.firstName +" prefers "+ pref );
    }); // fat arrows don't hijack this
```

SPREAD OPERATOR

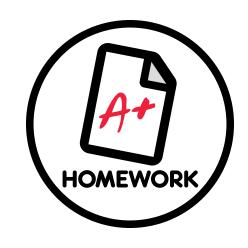
```
const cat = {
  legs: 4,
  sound: 'meow'
};
const dog = {
  ...cat, // The spread operator only makes a shallow copy!
  sound: 'woof' // This overwrites the cat sound
  //if we used ...cat here instead, it would overwrite sound
};
console.log(dog); // => { legs: 4, sounds: 'woof' }
```

REST PARAMETERS

```
function sum(...allArguments) {
 return allArguments.reduce((previous, current) => {
   return previous + current;
 });
console.log(sum(1, 2, 3, 4));
// output: 10
function f(a, b, ...theArgs) {
 // The first and second arguments are addressable as a and b
 // The remaining are addressable as an array called the Args
```

HOMEWORK

- Submit your HTML/CSS via Slack
- There are two weeks to finish the Javascript for your final projects



- Make an appointment with me to review your project and course progress
 - Tues/Thurs: Day or Evening by Skype
 - Fri: 9:00 AM 6:00 PM on campus
 - Sat: 10:00 AM 2:00 PM on campus
 - Sat/Sun: Day or Evening by Skype

EXIT SURVEYhttps://goo.gl/EB4XFw

EXIT SURVEYhttps://goo.gl/EB4XFw

GO BUILD AWESOME THINGS!