ES6 AND ADVANCED JAVASCRIPT

SCOPES

1. Block Scopes (aka lexical scopes)

```
{
 // anything within { } that isn't a function
}

if (isPhiladelphia) {
 var alwaysSunny = true;
}
```

2. Function Scopes

```
function doFunctionThings() {
 // this block is a function block
}
```

`var` only respects function scopes

LET AND CONST

- You've seen `var` but now we also have `let` and `const`
- Also used to store data

`let`

- Unlike `var`, `let` has scope.

Using `var`

```
for (var i = 0; i < 10; i++) {
  console.log(i); // changes from 0 to 9 as it goes through the loop
}
console.log(i); // 10
```

Using `let`

```
for (let i = 0; i < 10; i++) {
  console.log(i); // changes from 0 to 9 as it goes through the loop
}

console.log(i); // i is not defined
```

ACTIVITY: `let`

Modify this code so that 'a' is undefined outside the if statement.

```
var condition = true;

if (condition) {
  var a = 'Zeta';
  console.log(a); // Zeta
}

console.log(a); // This should be undefined
```

const

- for values that should never change or that you don't want changed
- Example: apply a discount. Don't allow the discount to be changed.

```
const discount = 3.4;
discount = 5.6 // this will throw an error
```

const ARRAY

```
const LANGUAGES = ['JavaScript', 'Ruby', 'Python', 'Go'];

LANGUAGES = "JavaScript"; // throws error

LANGUAGES.push('Java'); // This works!
```

ACTIVITY: `const`

```
const arr = [1, 2, 3];
const obj = { id: 0, name: 'Alpha' };

// Modify the array's contents; no reassignment
// Your Code Here

// Modify the object's contents; no reassignment
// Your Code Here
```

ACTIVITY: `const` AND `let`

The below code should output 0 1 2, but is outputting 3 3 3. Can you solve the issue using block scoping? Can you replace all the `var` with `const` or `let`?

```
var funcs = [];

for (var i = 0; i < 3; i++) {
  funcs.push(function() {
    console.log(i);
  });
}

for (var j = 0; j < funcs.length; j++) {
  funcs[j](); // should output 0, then 1, then 2
}
```

ARROW FUNCTION

- Newer syntax for functions

```
// Old Syntax
function oldOne() {
  console.log("Hello world!");
}

// New Syntax
var newOne = () => {
  console.log("Hello World!");
}
```

```
let newOneWithParameters = (a, b) => {
  return a + b;
}

newOneWithParameters(10, 20); // 30
```

ARROW FUNCTION EXAMPLE

```
let newOneWithParameters = (a, b) => {
  return a + b;
}

newOneWithParameters(10, 20); // 30
```

One step further:

```
let newOneWithParameters = (a, b) => a + b

newOneWithParameters(10, 20); // 30
```

What if you don't have parameters?

```
let double = a => 2 * a

double(2); // 4
```

ARROW FUNCTION STEPS

```
var getMessage = function(name) {
  return 'Hello ' + name + '!';
}
```

1. Remove the word `function`
2. Place a fat arrow (=>) after the parameters
3. If there's only 1 parameter, remove the surrounding parentheses ()
4. If there's only 1 expression in the function body, remove {} and return

```
const getMessage = name => 'Hello ' + name + '!';
```

ACTIVITY: REFACTOR CODE

Refactor the following code samples to use arrow functions.

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

console.log(sum(3, 4)); // 7
```

```
const fibonacci = function(n) {
  if (n < 3) {
    return 1;
  }
  return fibonacci(n - 1) + fibonacci(n - 2);
}

console.log(fibonacci(9)); // 34
```

DEFAULT PARAMETERS

```
let addNumbers = (a, b = 10) => {
  return a + b;
}

addNumbers(20); // 30

addNumbers(20, 30); // 50
```

ACTIVITY: DEFAULT PARAMETERS

INSTRUCTIONS: Write a function called multiply that accepts two parameters. The second parameter should have a default value of 1.

BONUS: Throw an error if the first parameter hasn't been entered.

```
// Modify this function
function multiply() {

}

console.log(multiply(5, 4));
console.log(multiply(4));
console.log(multiply());
```

FOR OF LOOP

- iterates through a list of elements and returns the elements one by one

```
let array = [2, 3, 4, 1];

for (let value of array) {
  console.log(value);
}

// Output:
// 2
// 3
// 4
// 1
```

SPREAD/REST OPERATOR ...

- Converts a list of elements to an array and vice versa

```
let createArray = (...array) => {
  console.log(array);
}

createArray(10, 20, 40, 60, 90); // [10, 20, 40, 60, 90]
```

```
let createListFromArray = (...list) => {
  console.log(list);
}

createListFromArray([10, 20, 40, 60, 90]); // 10, 20, 40, 60, 90
```

ACTIVITY: ...REST

INSTRUCTIONS: Refactor the following to use the rest operator. There should be no restriction on how many numbers the function accepts.

```
function product(x, y, z) {
  var numbers = [x, y, z];
  return numbers.reduce((currentProduct, number) => currentProduct * number, 1);
}

console.log(product(3, 4, 5)); // 60
console.log(product(3, 4, 5, 6)); // 360
console.log(product(1, 2, 3, 4, 5, 6)); // 720
```

ACTIVITY: ...SPREAD

INSTRUCTIONS: Refactor this code to use the spread operator instead of the concat method).

```
function join(array1, array2) {
  return array1.concat(array2);
}

console.log(join([1, 2, 3], [4, 5, 6])); // [1, 2, 3, 4, 5, 6]
```

TEMPLATE LITERALS

```
const student = {
  name: 'Jane Doe',
  city: 'Calgary'
};

let message = 'Hello ' + student.name + ' from ' + student.city;
```

Now:

```
let message = `Hello ${student.name} from ${student.city}`;
```

ACTIVITY: TEMPLATE LITERALS

- Create a div with an id of "target" in your index.html page.
- Refactor this code to use template strings.

```
const user = {
  name: 'Cody',
  loginCount: 1,
  goldStatus: true
};

const className = 'container';
const html = '<div class="' + className + '">'
        + '<h2>Welcome' + (user.goldStatus ? ' to our gold status member, ' : ', ') +
user.name + '!' + '</h2>'
        + '<p>Today is ' + new Date() + '</p>'
        + '<p>You have logged in ' + ++user.loginCount + ' times.</p>'
      + '</div>';

document.getElementById('target').innerHTML = html;
```

DESTRUCTURING AN ARRAY

- Data can be extracted from arrays and objects into distinct variables using destructuring.

```
const points = [20, 30, 40];

const [x, y, z] = points;

console.log(x, y, z); // 20 30 40
```

Ignore a value:

```
const [x, , z] = points;
```

DESTRUCTURING AN OBJECT

```
const car = {
  type: 'Toyota',
  color: 'silver',
  model: 2007
};

const {type, color, model} = car;

console.log(type, color, model); // Toyota silver 2007
```

```
const {color} = car;
console.log(color); // silver
```

ACTIVITY: DESTRUCTURE AN OBJECT

INSTRUCTIONS: Can you get the body of isActiveEngineer() down to two lines using destructuring?

```
const myEmployee = {
  active: true,
  department: 'Engineering'
};

function isActiveEngineer(employee) {
  const active = employee.active;
  const department = employee.department;
  return department === 'Engineering' && active;
}

console.log(`Is myEmployee active? ${isActiveEngineer(myEmployee)}`);
```

OBJECT LITERAL SHORTHAND

```
let type = 'Toyota';
let color = 'silver';
let model = 2007;

const car = {
  type, // instead of type: type
  color, // instead of color: color
  model // instead of model: model
};

console.log(car);
//output { type: 'Toyota', color: 'Silver', model: 2007 }
```

ARRAY HELPERS

Tons of new methods that can be applied to arrays

Array Helper	Use it to...	Returns
`arr.forEach((element, index, array) => { });`	loop through elements in an array	`undefined`
`arr.every((element, index, array) => { });`	check if every element passes a test	`true||false`
`arr.some((element, index, array) => { });`	check if some (at least 1) `arr` elements pass a test	`true||false`
`arr.filter((element, index, array) => { });`	create a new array by filtering the original array elements	`Array`
`arr.map((element, index, array) => { });`	create a new array by modifying the original array elements	`Array`
`arr.find((element, index, array) => { });`	find the first element that passes a test	1 element
`arr.findIndex((element, index, array) => { });`	find the index of the first element that passes a test	1 element
`arr.reduce((calculatedValue, element, index, array) => { }, initialValue);`	pass in an `initialValue` and modify it according to the current `element` value	1 reduced value

forEach

```
arr.forEach((element, index, array) => {});
```

- loop through elements in an array

Original

```
var names = ['Morgan', 'Taylor', 'Lesley'];

for (var i = 0; i < names.length; i++) {
  console.log(names[i]);
}
```

forEach

```
const names = ['Morgan', 'Taylor', 'Lesley'];

names.forEach((name, index, array) => {
  console.log(name);
});
```

ACTIVITY: `forEach`

INSTRUCTIONS: Each trip has a rate (mph) and time (hours) which we are using to calculate distance (mph * hours). Refactor this code to use the forEach helper instead of a for loop.

```
const trips = [
  { mph: 10, hours: 3 },
  { mph: 30, hours: 2 },
  { mph: 25, hours: 4 }
];

for (let i = 0; i < trips.length; i++) {
  console.log(trips[i].mph * trips[i].hours);
}
```

every

```
arr.every((element, index, array) => {});
```

- check if every element passes a test

```
const students = [
  { name: 'Morgan', present: true },
  { name: 'Sam', present: false },
  { name: 'Taylor', present: true }
];

const allPresent = students.every(student => student.present);
console.log(allPresent); // false
```

`some`

```
arr.some((element, index, array) => {});
```

- check to see if `some` (at least 1) `arr` elements pass a test

```
const students = [
  { name: 'Morgan', present: true },
  { name: 'Sam', present: false },
  { name: 'Taylor', present: true }
];

const somePresent = students.some(student => student.present);
console.log(somePresent); // true
```

ACTIVITY: `every` AND `some`

INSTRUCTIONS: You're managing IT inventory and want to check if all of your available laptops have at least 16 GB of RAM. If not, do some of them have at least 16 GB of RAM?

```
const availableLaptops = [
  { name: 'MacBook', RAM: 8 },
  { name: 'Asus', RAM: 32 },
  { name: 'Lenovo', RAM: 16 },
  { name: 'HP', RAM: 4 }
];

// your code here
```

filter

```
arr.filter((element, index, array) => {});
```

- create a new array by filtering the original array elements

```
const users = [
  { username: 'ryan10', active: true },
  { username: 'morgan', active: false }
];

const activeUsers = users.filter(user => user.active);
console.log(activeUsers);
```

ACTIVITY: `filter`

INSTRUCTIONS: Use filter to create an array of produce that cost less than $5

```
// Create an array of groceries
const groceries = [
  { name: 'bananas', aisle: 'produce', price: 2 },
  { name: 'flour', aisle: 'baking', price: 5 },
  { name: 'avocados', aisle: 'produce', price: 4},
  { name: 'lettuce', aisle: 'produce', price: 5 },
  { name: 'olive oil', aisle: 'baking', price: 10 },
  { name: 'shampoo', aisle: 'beauty', price: 12 }
];

const produceLessThan5 = []; // your code here

// Output the produce
console.log(produceLessThan5);
```

map

```
arr.map((element, index, array) => {});
```

- create a new array by modifying the original array elements

```
const numbers = [2, 6, 10];

const halvedNumbers = numbers.map(number => number / 2);

console.log(halvedNumbers); // 1, 3, 5
```

ACTIVITY: `map`

INSTRUCTIONS: Use the map() method to create an array called 'speeds' based on the 'trips' array. Speed is calculated as miles / hours.

```
const trips = [
  { miles: 5, hours: 1},
  { miles: 6, hours: 0.5},
  { miles: 4, hours: 2 }
];

const speeds = []; // your code here

// Output the speeds
console.log(speeds);
```

find

```
arr.find((element, index, array) => {});
```

- find the first element that passes a test

```
const jobs = [
  { title: 'Electrician' },
  { title: 'Developer' },
  { title: 'Barista' }
];

const devJob = jobs.find(job => job.title === 'Developer');
console.log(devJob); // { title: 'Developer' }
```

ACTIVITY: `find`

INSTRUCTIONS: We need to console.log the title and body of the post with ID 29. Use the find() method to set the value of 'currentPost' to the object with ID 29. If everything worked, you should see the title and body in your console.

```
// An array of posts
const posts = [
  { id: 23, title: 'Becoming an ES6 Wizard', body: 'Ille vivere. Ut ad te ...' },
  { id: 29, title: 'JavaScript Pro Tips', body: 'Pellentesque euismod ...' },
  { id: 31, title: 'Acing your next JS interview', body: 'Lorem ipsum dolor sit
amet...'}
];

// Find the post with ID 29
const currentId = 29;
const currentPost = ''; // your code here

console.log(currentPost.title);
console.log(currentPost.body);
```

findIndex

```
arr.findIndex((element, index, array) => {});
```

- find the index of the first element that passes a test

```
const numbers = [ 25, 30, 35, 40, 45 ];
const firstIndex = numbers.findIndex(number => number > 33);
console.log(firstIndex); // 2 (the location of 35)
```

`reduce`

```
arr.reduce((calculatedValue, element, index, array) => {}, initialValue);
```

- pass in an `initialValue` and modify it according to the current `element` value

```
const numbers = [2, 6, 10];

const sum = numbers.reduce(function(currentSum, number) {
  return currentSum + number;
}, 0);

console.log(sum); // 18
```

ACTIVITY: `reduce`

INSTRUCTIONS: Use reduce() to find the sum of the elements in the transactions array. Remember to return the current calculated value (the first parameter in the iterator function) every iteration.

```
const transactions = [ 5, 10, 15 ];

const sum = 0;

console.log(sum); // 30
```

ACTIVITY: QUIZ

Which array helper would be most suitable for each situation? (forEach, every, some, filter, map, find, or reduce)

1. You need to check if all elements in an array are integers.
2. You have an array of blog posts and need to output each post on the page.
3. You need to check if an array has at least one element that isn't null.
4. You have an array of expenses and need to calculate your total expenses last month.
5. You need a new array to store the square root of each item in an existing array.
6. You have an array of computers, but only need computers with more than 16 GB of RAM.
7. You have an array of blog posts, and need to retrieve one post by its ID.