Intro to JavaScript

Watch the Recording!

Schedule

1. Learning Objectives (5 minutes)
2. Warm-Up: Observations (5 minutes)
3. JavaScript Fundamentals & Arrow Functions (20 minutes)
4. Activity: Pair Programming (15 minutes)
5. BREAK (10 minutes)
6. Intro to Promises (20 minutes)
7. Work Time: Codecademy JS (30 minutes)
8. Wrap-Up

Learning Objectives

By the end of this lesson, students will be able to…

1. Use basic JavaScript constructs such as functions, conditionals, loops, and objects.
2. Read, understand, and use arrow functions to write shorter, more concise code.
3. Create and resolve Promises in order to execute code asynchronously.

Warm-Up: Observations

List 3 things that are different in JavaScript than what you are used to in Python. Submit your list in the Zoom chat.

TT: JavaScript fundamentals [30 minutes]

Semicolons

JavaScript has **optional semicolons** which can be used after statement of code. You will see them used in tutorials and assig
you do not need to use them yourself.

Variables: const, let, var

Use `const` if you are declaring a variable which will not be reassigned:

Click to Copy

```
const myVar = 'hello';
myVar = 4;
TypeError: Assignment to constant variable.
```

We can, however, still *modify* the value of an object or list contained in a const variable; we just can't assign it to a *new* object

```
const myList = [1, 2, 3, 4];
myList.push(5); // contains [1, 2, 3, 4, 5]
```

Use `let` if you are declaring a variable which will be reassigned:

```
let myVar = 'hello';
myVar = 5;
```

In general, do not use `var` to assign variables. *Read **here** for an example of a bug caused by using var (and easily avoided w*

Scoping

`const` and `let` both have “block” scoping - that is, a variable assigned in a block is not available to be used outside of the encapsulate our variable declarations, but it does lead to some interesting errors.

```
if (4 === 4) {
    let result = "four equals four";
}
console.log(result);
```

Result: `ReferenceError: result is not defined`

Conditionals

A **conditional** in JavaScript looks like:

```
if (condition) {
    // do something
}
```

where the *condition* evaluates to a boolean (true or false).

Note: When comparing for equality, always use the `===` operator, not `==` . This is because `===` uses **strict** equality, which i

For Loops

A typical for loop will use a **counter** variable to run a specific number of times.

```
const myList = [ ... ]
for (let i = 0; i < myList.length; i += 1) {
    console.log(myList[i])
}
```

Follow-up questions: How do you…

- Traverse the list backwards?
- Traverse every other element?

Functions

To write a function, use the `function` keyword:

```
function sayHello(name) {
    console.log(`Hello, ${name}!`)
}
```

Objects

Think of JavaScript objects (aka JSON) as collections of key-value pairs, where the values can either be *primitives* (boolean, nu
pairs.

```
const userInfo = {
    name: 'Ada',
    favoriteColor: 'blue',
    address: {
        street: '851 California St',
        city: 'San Francisco',
        state: 'CA'
    }
}
```

We can access the contents like this:

```
const name = userInfo.name
```

Arrow Functions

Arrow functions were added to the core JavaScript syntax as part of **ES6**. They work (almost) exactly like regular functions, bu

```
// SHORTEST VERSION
const doubleNum = num => num * 2;

// MORE VERBOSE VERSION
const tripleNum = (num) => {
    return num * 3;
}

doubleNum(6) // 12
tripleNum(6) // 18
```

The above two examples are exactly the same as if I had written:

```
const doubleNum = function(num) {
    return num * 2;
}
// OR
function doubleNum(num) {
    return num * 2;
}
```

Arrow functions are just **“syntactic sugar”** for concepts that we already know!

Activity: Pair Programming (15 minutes)

Choose pairs randomly. With your partner, go to the **BEW 1.3 Repl.It classroom** and follow the instructions to complete the as

Break (10 minutes)

Intro to Promises

Why do we care about asynchronous code?

JavaScript is a *single-threaded* language, meaning it can only do one thing at any given time. Think of it like when you are worl chopping vegetables, washing dishes, or putting away groceries.

But, sometimes we need to call *other* libraries’ functions, and those take some time to return. Think of those like a dishwasher them to finish before going on to another task - you want to take advantage of that wait time to do something else.

But how do we know when that external task is finished? Some languages use *event listeners*, which “listen” for a particular al machine to play a song to tell you that it’s done). JavaScript, however, uses **Promises** and **callbacks**, which let us *specify wha*

Working with a Promise return value

Calling a regular function is like washing the dishes by hand: the program can’t work on anything else until it’s done.

```
function wash_dishes_by_hand() {
    console.log('Washing plates...')
    console.log('Washing bowls...')
    console.log('Washing spoons...')
    return 'Done!'
}

wash_dishes_by_hand(); // prints 'Done!'
```

Using a Promise is like running the dishwasher: the program *can* work on other things while it runs. *As soon as we make the p*

```
const washDishesExecutor = (resolve, reject) => {
    // ... washing dishes (this step takes some time) ...
    if (dishes_are_clean) {
        resolve('Success! Dishes are done!')
    } else {
        reject('Error! No soap!')
    }
}
const washDishesPromise = new Promise(washDishesExecutor);
```

But how do we know when it’s done? We can *resolve* a promise to specify what to do when it finishes!

```
washDishesPromise.then((result) => {
    console.log(result) // prints 'Success! Dishes are done!'
}).catch((err) => {
    console.log(err) // prints 'Error! No soap!'
});
```

Try out these examples on your own and see if you can experiment with using promises!

Work Time: Codecademy

Work on **Codecademy JS** parts 12 and 13.

Wrap-Up

Fill out our **Vibe Check form** with any feedback you have for the class.

Your *two* homework assignments (due by next class) are:

1. **Codecademy JS** parts 12 and 13
2. **Promise Challenges**

Make sure to submit your work through **Gradescope**.