JavaScript Loop, Array, and Functions

Harry J. Wang, Ph.D.

University of Delaware

References

- Object-oriented JavaScript for beginners tutorial: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Object-oriented_JS
- Part of the slides are reviews for this course: https://www.codecademy.com/learn/learn-javascript

See Additional Array Methods

Arrays

- Arrays store **ordered list** of data
- JavaScript is zero-indexed.

```
let bucket_list = ['Build a house', 'Learn to paint', 'buy a motocycle'];
console.log(bucket_list);
console.log(bucket_list[2]);
```

- length property can be used to find how many items are stored inside of an array.
- .push()/.pop() allows us to add/remove items to **the end** of an array

```
console.log(bucket_list.length);  // 3
bucket_list[2] = "Visit Japan";

bucket_list.push('Climb a mountain');
console.log(bucket_list.length);  // 4

bucket_list.pop();
console.log(bucket_list.length);  // 3
console.log(bucket_list.indexOf("Learn to paint"));  // 1
```

For Loop (number of loops known)

- Simple for loop

```
let bucket_list = ['Build a house', 'Learn to paint', 'buy a motocycle'];
console.log("My bucket list has the following items:");
for (let i = 0; i < bucket_list.length; i++) {
  console.log( (i + 1) + ": " +bucket_list[i]);
}
```

- Nested loop

```
let imdb_movie_list = [
  "The Shawshank Redemption",
  "The Godfather",
  "The Godfather: Part II",
  "The Dark Knight",
  "12 Angry Men"
];

let ranker_movie_list = [
  "The Godfather",
  "The Shawshank Redemption",
  "Pulp Fiction",
  "Star Wars",
  "Forrest Gump",
  "The Dark Knight"
];
```

```
console.log("matched movies:");
for (let i = 0; i < imdb_movie_list.length; i++) {
  for (let j=0; j < ranker_movie_list.length; j++) {
    if (imdb_movie_list[i] == ranker_movie_list[j]) {
      console.log(imdb_movie_list[i]);
    }
  }
}
```

```
matched movies:
The Shawshank Redemption
The Godfather
The Dark Knight
```

While Loop (number of loops unknown)

```
while (condition) {
  // code block that loops until
condition is false
}
```

```
let balance = 100;
let years = 0;
while (balance < 1000000) {
  balance += balance * 0.05;
  years += 1;
}
console.log("You will be a millinaire in " + years + "!");
```

```
You will be a millinaire in 189 years !
```

https://www.w3schools.com/js/js_functions.asp

Function

- A JS function is a block of code designed to perform a particular task.
- A JS function is executed when "something" invokes it (calls it).
- **Function Declaration**: the **function** keyword, followed by a **name**, followed by parentheses **()**, which may include parameter names separated by commas.

```
function name(parameter1, parameter2, parameter3) {
    code to be executed
}
```

When **return statement** is reached, the function will stop executing.

```
// Function returns the product of a and b
function get_production(a, b) {
return a * b;
}

// Function is called, return value will end up in x
let x = get_production(3,4);
console.log(x)
```

Why Functions?

- Functions are created to reuse code, e.g., use the same code many times with different arguments to produce different results.

```
// Convert Fahrenheit to Celsius
function get_celsius(fahrenheit) {
  return (5/9) * (fahrenheit-32);
}
console.log(get_celsius(104))
console.log(get_celsius(89))
```

```
40
31.666666666666668
```

Global vs. Block Scope

- A **block** refers to the {} curly braces, which is an important structural marker for our code.
- **Global Scope**: variables are declared outside of any block.
- **Block scope**: a variable defined in the block is only accessible within the curly braces.

```
let status = "clear" // global scope
function get_sky_status() {
  let color = 'blue'; // block scope
  console.log(color); // blue
  console.log(status); // clear
};

get_sky_status(); // blue
console.log(status); // clear
console.log(color); // undefined error
```

```
▶Uncaught ReferenceError: color is not defined
    at js.html:200
```

Link External JS Files

- To run the same JavaScript on several web pages, an external JavaScript file should be created with a .js extension and linked in the HTML file.
- The external script file cannot contain the <script> tag.

```
<script src="my-scripts.js"></script>
```

In-class Exercise

- Implement some loop example in the lecture
- Change the grade calculator into a function declaration and an arrow function
- Use block scope variables

Questions?

Iterators

- Iterators are array methods that loop over an array and select elements that meet certain criteria.
- .forEach() loop over a array without changing its elements.
- Try to use arrow function

```
// iterator function
imdb_movie_list.forEach(function(movie){
  console.log(" - " + movie);
});

// iterator arrow function
imdb_movie_list.forEach(movie => console.log(" - " + movie));
```

.map() and .filter()

- .map() returns a **new array** with elements that have been modified by the code in its block
- .filter() returns a **new array** with certain elements from the original array that evaluate to truth based on conditions written in the block of the method.

```
// map
let my_movie_list = imdb_movie_list.map(movie => "The Godfather");
console.log(my_movie_list);

// return movie with name starts with "The"
// .slice(a, b): a begin, b up to not including
let the_movie_list = imdb_movie_list.filter(movie => movie.slice(0,3) == "The");
console.log(the_movie_list);
```

- See other iterators

Objects

- JavaScript *objects* are containers that can store **data** and **functions**.
- The data in an object is **not ordered**, which can only be accessed by using its associated *key*.
- Create an object with **key-value** pairs:

```
let player1 = {
  name: "Stephen Curry",
  dob: "March 14, 1988",
  height: 1.91,
  weight: 86,
  shoot(){
    return "3-pointer";
  }
};
```

```
let player2 = {
  name: "LeBron James",
  dob: "December 30, 1984",
  height: 2.03,
  weight: 118,
  shoot(){
    return "dunk";
  }
};
```

```
console.log(player1.name);  // dot notation
console.log(player2["name"]); // bracket notation
console.log(player2.weight - player1.weight);
console.log(player1.shoot());
console.log(player2.shoot());
```

```
Stephen Curry
LeBron James
32
3-pointer
dunk
```

this keyword

- In Javascript, *this* refers to the object we call it inside.
- Add object property value on the fly

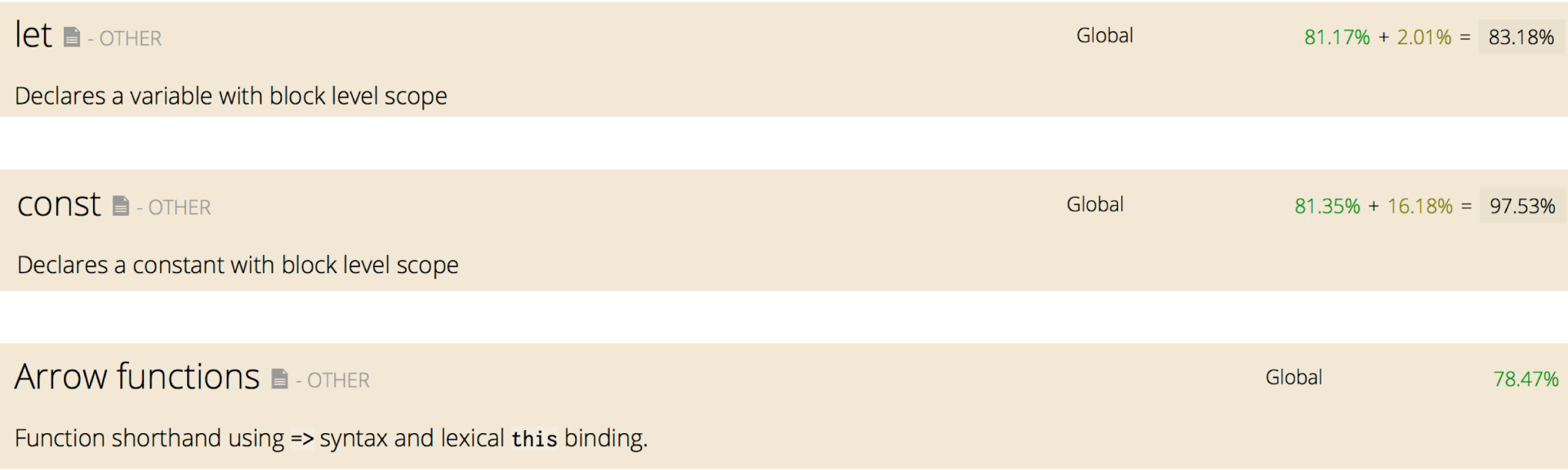
```
let player2 = {
  name: "LeBron James",
  dob: "December 30, 1984",
  height: 2.03,
  weight: 118,
  shoot(){
    return "dunk";
  },
  info(){
    return `${this.name} is ${this.height}m and ${this.weight}kg.`;
  }
};
```

```
console.log(player2.info());
player2.draft_year = 2003;
console.log(player2.draft_year);
```

```
LeBron James is 2.03m and 118kg.
2003
```

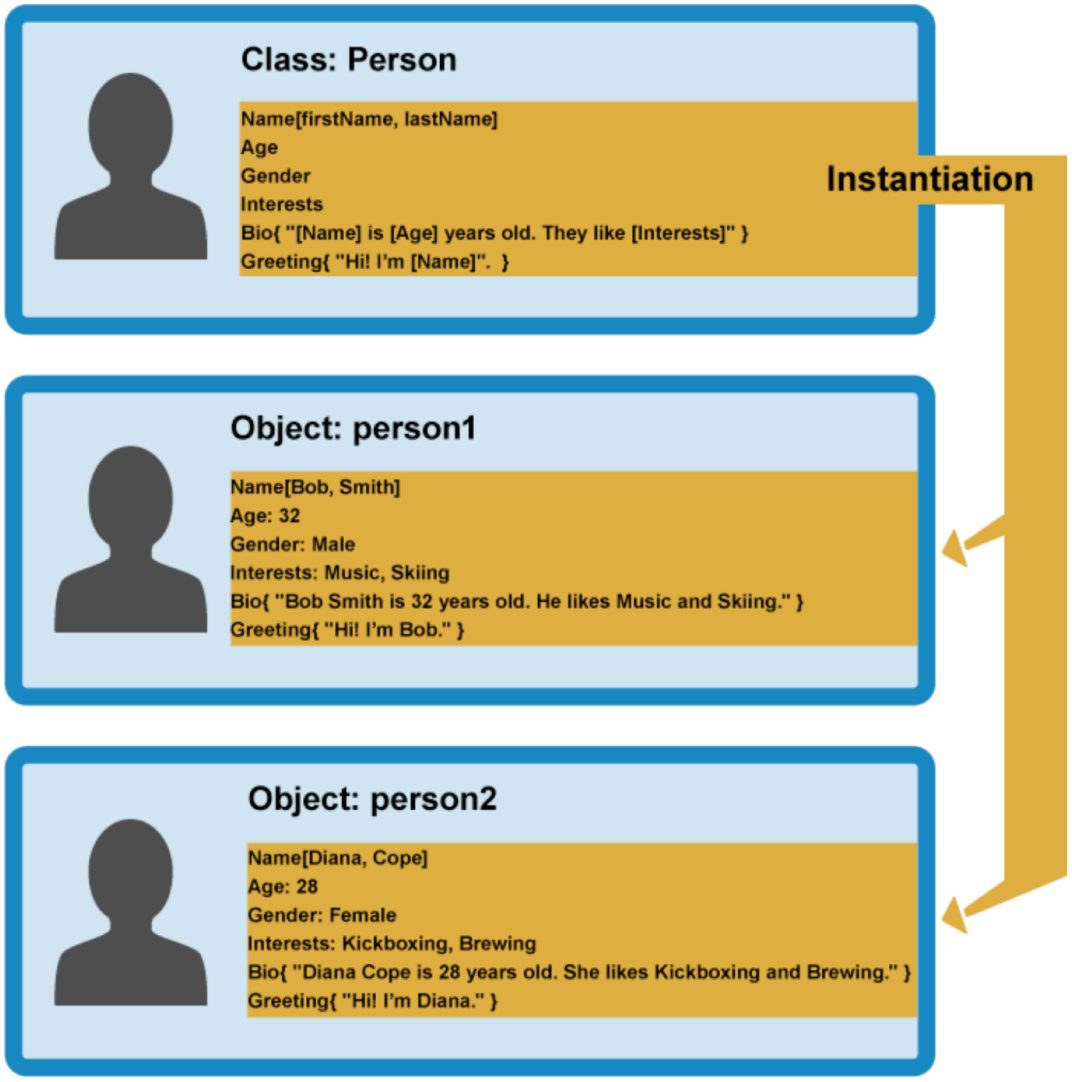
Browser Compatibility (caniuse.com)

- let, const vs. var
- Arrow function
- Spring concatenation
- **Sections on NPM is optional**



Object-Oriented Programming (OOP)

- OOP uses objects to model real world things that we want to represent inside our programs.
- An object can contain related data (**properties**) and code (**methods**), which represent information about the thing and its functionality or behavior.
- When an object instance is created from a class, the class's **constructor function** is run to create it.
- This process of creating an object instance from a class is called **instantiation**.



Constructor Function

- Unlike many OO languages, JavaScript doesn't have a class statement for creating classes.
- JS uses special functions called **constructor functions** to define objects and their features.

```
// constructor function
function Person(name) {
  this.name = name;
  this.greeting = function() {
    console.log("Hi! My name is " + this.name + ".");
  };
}
```

```
// create object instances
var person1 = new Person('Bob');
var person2 = new Person('Sarah');
console.log(person1.name)
person1.greeting()
console.log(person2.name)
person2.greeting()

// without (), the definition is returned
console.log(person1.greeting)
```

```
Bob
Hi! My name is Bob.
Sarah
Hi! My name is Sarah.
f () {
        console.log("Hi! My name is " + this.name + ".");
      }
```

Function Expression

- A function expression is similar to function declaration, with the exception that identifier can be omitted, creating an **anonymous function**.
- An **arrow function expression** has a shorter syntax than a function declaration.

```
// Anonymous function expression
const square = function (number) {
  return number * number;
};

console.log(square(5));  // 25
```

```
// Arrow Function is a shorter way
const square = (number) => {
    return number * number;
};

console.log(square(5));
// Output: 25.
```

https://www.w3schools.com/js/js_hoisting.asp

JS Hoisting

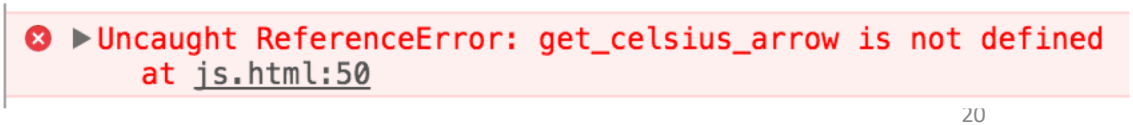
- Hoisting is JavaScript's default behavior of moving all declarations to the top of the current scope (to the top of the current script or the current function).
- Arrow functions are not hoisted to the top of the scope.

```
console.log(get_celsius(104))
console.log(get_celsius(89))

// Convert Fahrenheit to Celsius
// function declaration is hoisted
// to the top of the scope
function get_celsius(fahrenheit) {
  return (5/9) * (fahrenheit-32);
}
```

```
// this won't work
console.log(get_celsius_arrow(104))
console.log(get_celsius_arrow(89))

// Convert Fahrenheit to Celsius
// function expression - not hoisted
const get_celsius_arrow = (fahrenheit) => {
  return (5/9) * (fahrenheit-32);
}
```



Arrow Function (cont.)

- Arrow functions cannot be used as constructors and will throw an error when used with *new* keyword.

```
// constructor function
function Person(name) {
  this.name = name;
  this.greeting = function() {
    console.log("Hi! My name is " + this.name + ".");
  };
}
```

```
// arrow function cannot be used as constructor
const ArrowPerson = (name) => {
  this.name = name;
  this.greeting = () => {
    console.log("Hi! My name is " + this.name + ".");
  };
}

// this won't work
let arrow_person = new ArrowPerson('Harry');
```

```
▶Uncaught TypeError: ArrowPerson is not a constructor
    at js.html:101
```

Condensed Arrow Functions

- **No parentheses**: Functions that take **a single parameter** should not use parentheses.
- **No return** : A function composed of **a sole single-line block** is automatically returned. The contents of the block should immediately follow the arrow => and the return keyword can be removed. This is referred to as implicit return.
- **No Brackets**: A function composed of **a sole single-line block** does not need brackets.

```
// normal arrow functions
const multiplyByNineFifths = (celsius) => {
  return celsius * (9/5);
};

const getFahrenheit = (celsius) => {
  return multiplyByNineFifths(celsius) + 32;
};
```

```
// condensed arrow functions
const multiplyByNineFifths = celsius => celsius * (9/5);

const getFahrenheit = celsius => multiplyByNineFifths(celsius) + 32;

console.log('The temperature is ' + getFahrenheit(15) + '°F');
```