# Collections and Loops

**David Tucker**

CTO Consultant

@_davidtucker_ | davidtucker.net

# Array

An Array in JavaScript is an ordered collection of items. It has a length property and methods for manipulating the list. Arrays should be used for collections where the order matters.

# Using Arrays

arrays.js

```
// Creating and populating an array
let departments = [ "Marketing", "Engineering" ];

// Accessing array items
console.log(`First Item: ${departments[0]}`);

// Checking the array length
console.log(`Length: ${departments.length}`); // 2

// Add item to the array
departments.push("Human Resources");

// Updated length
console.log(`Length: ${departments.length}`); // 3
```

# Map

A Map is a JavaScript collection type that allows you to use any data type as your key. It has a size property and methods for manipulating the collection.

# Using Maps

maps.js

```
// Creating a Map
let accessCodes = new Map();

// Populating a Map
accessCodes.set("employee1234", "8303");
accessCodes.set("employee1235", "1111");
console.log(accessCodes.size); // 2

// Retrieving values from a Map
let code = accessCodes.get("employee1234");
console.log(code); // 8303

// Delete values from a Map
accessCodes.delete("employee1235");
console.log(accessCodes.size); // 1
```

# Set

A Set in JavaScript is a collection type that allows for a unique set of values. It has a size property and methods for manipulating the set.

# Using Sets

sets.js

```
// Creating a Set
let agileTeam = new Set();

// Populating a Set
agileTeam.add("employee1234");
agileTeam.add("employee1235");
console.log(agileTeam.size); // 2

// Trying to re-add a value
agileTeam.add("employee1235");
console.log(agileTeam.size); // 2 (unchanged)
```

# Arrays

# Map and Set

# Loops

# Loop

In JavaScript, a loop executes a block of statements until a specific condition is met. One such condition could be reaching the end of a collection, but it also could be any other conditional you could think of.

# While Loops

while.js

```
// While Loop
let x = 10;
while(     )



// 10, 9, 8, 7, 6, 5, 4, 3, 2, 1
```

# Do While Loops

do-while.js

```
// Do-while Loop
let x = 10;
do {
  console.log(x);
  --x;
} while(x > 0);
// 10, 9, 8, 7, 6, 5, 4, 3, 2, 1
```

# For Loops

for.js

```
// For Loop
let departments = ["Marketing", "Engineering", "HR"];
for(                                              ) {

}

// For...of Loop
for(                          ) {

}

// For...in Loop
let obj = { firstName: "David", lastName: "Tucker" };
for(                  ) {

}
```

The break keyword allows us to terminate the loop and move to the next statement in our code.

The continue keyword stops the current execution of the loop and proceeds to the next iteration.

# While Loops

Up Next:

# For Loops

# For Loops

# Loop Control Flow

# Reading CLI Arguments

# Demo

Reading command line arguments passed to our application

Utilizing loops to display information for multiple employees