9.1 Loops

while loop

Three general-purpose looping constructs exist: `while`, `do-while`, and `for` loops. The ***while loop*** is a looping structure that checks if the loop's condition is true before executing the loop body, repeating until the condition is false. The ***loop body*** is the statements that a loop repeatedly executes.

Construct 9.1.1: while loop.

```
while (condition) {
   body
}
```

PARTICIPATION ACTIVITY 9.1.1: Executing a while loop.

Animation captions:

1. Assign i with 1.
2. 1 <= 4 is true, so the loop's body executes.
3. Output i to the console.
4. Increment i.
5. End of loop so go back to the top and re-evaluate the condition.
6. Keep executing loop until i <= 4 is false.

Developers must be careful to avoid writing infinite loops. An ***infinite loop*** is a loop that never stops executing. If the web browser is running JavaScript that contains an infinite loop, the web browser tab will cease to respond to user input.

Figure 9.1.1: JavaScript infinite loop with Chrome's "Page Unresponsive" message.

```
// Infinite loop!!!
while (true);
```

PARTICIPATION ACTIVITY 9.1.2: while loop.

1) What are the first and last numbers output by the code segment?

```
var c = 100;
while (c > 0) {
   console.log(c);
   c -= 10;
}
```

- ○ 100 and 0.
- ○ 90 and 0.
- ○ 100 and 10.

2) What condition makes the loop output the even numbers 2 through 20?


```
var c = 2;
while (_____) {
   console.log(c);
   c += 2;
}
```

- ○ c >= 20
- ○ c <= 20
- ○ c < 20

3) What is the value of `c` when the loop terminates?

```
var c = 10;
while (c <= 20); {
   console.log(c);
   c += 5;
}
```

- 25
- 20
- The loop never terminates.

4) What is `c` when the loop terminates?

```
var c = 10;
while (c <= 20)
   console.log(c);
   c += 5;
```

- 15
- 20
- The loop never terminates.

do-while loop

The ***do-while loop*** executes the loop body before checking the loop's condition to determine if the loop should execute again, repeating until the condition is false.

Construct 9.1.2: do-while loop.

```
do {
   body
} while (condition);
```

PARTICIPATION ACTIVITY 9.1.3: Executing a do-while loop.

Animation captions:

1. Assign i with 1.
2. do..while loop executes the loop body, evaluating the loop condition after the first execution.
3. The loop repeatedly executes until i <= 4 is false.

PARTICIPATION ACTIVITY 9.1.4: do-while loop.

1) What is the last number output to the console?

```
var c = 10;
do {
   console.log(c);
   c--;
} while (c >= 5);
```

Answer field

Check Show answer

2) Write a condition that executes the do-while loop as long as the user enters a negative number.

```
var num;
do {
   num = prompt("Enter a negative
number:");
} while (_____);
```

Answer field

Check Show answer

3) What is the last number output to the console?

```
var x = 1;
do {
   var y = 0;
   do {
      console.log(x + y);
      y++;
   } while (y < 3);
   x++;
} while (x < 4);
```

Answer field

Check Show answer

PARTICIPATION ACTIVITY 9.1.5: Practice with the while and do-while loops.

A given insect population doubles every week. If 2 insects exist on the first week, how many weeks will pass until the insect population exceeds 10,000 insects? Use a `while` loop to output the insect population each week until the population exceeds 10,000 insects.

Researchers have discovered that every 4 weeks a disease is killing 40% of the insect population after the population has reproduced. If 2 insects exist on the first week, how many weeks will pass until the insect population exceeds 10,000 insects? Write a second `do-while` loop that outputs the insect population each week until the population exceeds 10,000 insects. Decimal places will appear in the number of insects after removing 40% of the population on week 4.


```
1 // Write the while and do-while loops here!
2
```

Run JavaScript Reset code

Your console output

for loop

A for loop collects three parts — the loop variable initialization, loop condition, and loop variable update — all at the top of the loop. A ***for loop*** executes the initialization expression, evaluates the loop's condition, and executes the loop body if the condition is true. After the loop body executes, the final expression is evaluated, and the loop condition is checked to determine if the loop should execute again.

Construct 9.1.3: for loop.

```
for (initialization; condition; finalExpression) {
   body
}
```

PARTICIPATION ACTIVITY 9.1.6: Executing a for loop.

Animation captions:

1. for loop's initial expression assigns i with 1.
2. for loop's condition is then evaluated. 1 <= 4 is true, so the loop's statements are executed.
3. Output i to the console.
4. After the loop executes, the final expression is evaluated, which increments i.
5. Loop repeatedly executes until i <= 4 is false.

PARTICIPATION ACTIVITY 9.1.7: For loops.

1) Which loop always executes the loop body at least once?

   - `while`
   - `do-while`
   - `for`

2) What numbers are output by the code segment?

```
for (c = 5; c < 10; c += 2) {
   console.log(c);
}
```

- 5, 7, 9
- 5, 6, 7, 8, 9
- Infinite loop

3) Which condition causes the for loop to output the numbers 100 down to 50, inclusively?

```
for (c = 100; ______; c--) {
   console.log(c);
}
```

- `c < 50`
- `c > 50`
- `c >= 50`

4) What numbers are output by the code segment?

```
for (x = 1; x <= 3; x++) {
   for (y = 2; y <= 4; y++) {
      console.log(y);
   }
}
```

- 2, 3, 4
- 2, 3, 4, 2, 3, 4, 2, 3, 4
- 2, 3, 4, 5, 6, 7, 8, 9, 10

break and continue statements

*Two jump statements alter the normal execution of a loop. The **`break`** statement breaks out of a loop prematurely. The **`continue`** statement causes a loop to iterate again without executing the remaining statements in the loop.*

```
for (c = 1; c <= 5; c++) {
   if (c == 4) {
      break;   // Leave the loop
   }
   console.log(c);   // 1 2 3 (missing 4 and 5)
}

for (c = 1; c <= 5; c++) {
   if (c == 4) {
      continue;   // Iterate again immediately
   }
   console.log(c);   // 1 2 3 (missing 4) 5
}
```


*Some developers use **`break`** and **`continue`** to write short and concise code, but other developers avoid using jump statements on the grounds that jump statements may introduce subtle logic errors that are difficult to find. This material does not use jump statements.*

CHALLENGE ACTIVITY 9.1.1: Loops.

Start

Write a while loop that divides userNum by 2 while userNum is greater than 1, displaying the result after each division. Ex: For userNum = 40, output is: 20 10 5 2.5 1.25 0.625

```
1 var userNum = 40; // Code will be tested with values: 40, 4 and -5
2
3 /* Your solution goes here */
4
```


1 2 3

Check Next

9.2 Functions

A ***function*** is a named group of statements. JavaScript functions are declared with the `function` keyword. Functions may take any number of parameters and may return a single value using the `return` statement. A function that is missing a `return` statement returns `undefined`. Invoking a function's name, known as a function call, causes the function's statements to execute.

PARTICIPATION ACTIVITY 9.2.1: Declaring and calling functions.

Animation captions:

1. Declaring a function named displaySum with three parameters: x, y, and z.
2. Calling the displaySum function with arguments 2, 5, and 3 results in 10 being displayed in the console.
3. Declaring a function named findAverage with two parameters: a and b.
4. Calling the findAverage function with arguments 6 and 7 returns back 6.5.
5. Display the number returned by the average function.

Good practice is to use function names that contain a verb and noun. Ex: `display` is a vague function name, but `displayAverage` is better because `displayAverage` indicates what is being displayed.

Good practice is to use camel case for JavaScript function names, where the name starts with a lowercase letter and subsequent words begin with a capital letter.

PARTICIPATION ACTIVITY 9.2.2: Functions and return values.

1) Choose a better name for the function `test`.

```
function test(x, y) {
   if (x > y) {
      return x;
   }
   else {
      return y;
   }
}
```

- Max
- find_max
- findMax

2) What is output to the console?

```
console.log(min(5, 2));

function min(x, y) {
   if (x < y) {
      return x;
   }
   else {
      return y;
   }
}
```

- 2
- 5
- undefined

3) What is output to the console?

```
console.log(sayHello("Sam"));

function sayHello(name) {
   console.log("Hello, " + name);
}
```

- Hello, Sam
- Hello, Sam
  undefined
- undefined

4) What is output to the console?

```
sayHello("Sam");
sayHello("Juan", "Hola");

function sayHello(name, greeting)
{
   if (greeting === undefined) {
      greeting = "Hello";
   }
   console.log(greeting  + ", " +
name);
}
```


- `Hello, Sam`
  `Hello, Juan`
- `Hello, Sam`
  `Hola, Juan`
- `undefined`

PARTICIPATION ACTIVITY 9.2.3: Function practice.

The code below produces a 5 x 10 box of question marks. Convert the code into a function called `drawBox()` that takes three parameters:

1. `numRows` - The number of rows for the box.
2. `numCols` - The number of columns for the box.
3. `boxChar` - The character to use to create the box. If no argument is supplied, use "X".

Ex: `drawBox(5, 4, "!")` and `drawBox(2, 6)` should display the boxes pictured below.

```
!!!!
!!!!
!!!!
!!!!
!!!!
XXXXXX
XXXXXX
```

```
// Convert into a drawBox function
for (var r = 0; r < 5; r++) {
   var line = "";
   for (var c = 0; c < 10; c++) {
      line += "?";
   }
   console.log(line);
}

```


Run JavaScript Reset code

Your console output

Variables declared inside a function have ***local scope***, so only the function that defines a variable has access to the ***local variable***. Variables declared outside of a function have ***global scope***, and all functions have access to a ***global variable***.

Figure 9.2.1: Example with global variable.

```
// winner is a global variable
var winner = "Jill";
displayWinner();

function displayWinner() {
   // Displaying global variable
```

```
Jill
```

```
    console.log(winner);
}
```

Figure 9.2.2: Example with local variable.


```
function calculateTax(total) {
   // tax is a local variable
   var tax = total * 0.06;
   return tax;
}

var totalTax = calculateTax(10);

// Error because tax is not accessible outside calculateTax
console.log(tax);
```

```
Uncaught ReferenceError: tax is not defined
```

Variables assigned a value in a function but not declared in the function using the `var` become global variables. Good practice is to always declare variables used in functions with `var`, so the variables do not become global.

Figure 9.2.3: Example with accidental global variable.

```
function calculateTax(total) {
   // Missing "var" so tax becomes a global variable!
   tax = total * 0.06;
   return tax;
}

var totalTax = calculateTax(10);

// tax is accessible because tax is global
console.log(tax);
```

```
0.6
```

PARTICIPATION ACTIVITY 9.2.4: Variable scope and functions.

1) What is output to the console?

```
function addNumber(x) {
   sum += x;
}

var sum = 0;
addNumber(2);
addNumber(5);
console.log(sum);
```

Answer field

Check Show answer

2) What is output to the console?

```
function multiplyNumbers(x, y) {
   var answer = x * y;
   return answer;
}

var z = multiplyNumbers(2, 3);
console.log(answer);
```

Answer field

Check Show answer

3) What is output to the console?

```
function multiplyNumbers(x, y) {
   answer = x * y;
   return answer;
}

var z = multiplyNumbers(2, 3);
console.log(answer);
```

Answer field

Check Show answer

Global variables and the window object.

When running JavaScript code in a web browser, global variables are assigned as properties to the global ***window*** *object. Therefore, a global variable called* ***test*** *is accessible as* ***window.test****. Developers must be careful when assigning global variables because a global variable could replace an existing* ***window*** *property. Ex:* ***window.location*** *contains the URL the browser is displaying. Assigning* ***location = "Texas"*** *causes the web browser to attempt to load a web page with the URL "Texas", which does not exist.*

JavaScript functions may be assigned to a variable with a function expression. A ***function expression*** is identical to a function declaration, except the function name is omitted.

Figure 9.2.4: Assigning a function expression to a variable.


```
// Function name is omitted
var displaySum = function(x, y, z) {
   console.log(x + y + z);
}

// Function call
displaySum(2, 5, 3);
```

Unlike traditional functions, function expressions are not hoisted. ***Hoisting*** is JavaScript's behavior of moving variable declarations to the top of the current scope. Because function expressions are not hoisted, using a variable before the variable is assigned to a function expression causes an exception.

PARTICIPATION ACTIVITY 9.2.5: Hoisting variables and functions.

Animation captions:

1. Before any statements are executed, the declared function findLargest() is hoisted to the top.
2. findLargest() is called.
3. Before findLargest() executes, the function's local variables are hoisted to the top of the function.
4. Since x > y, largest is assigned x, and 5 is returned.
5. The function expression assigned to displaySum is not hoisted, so calling displaySum() produces an exception.

Functions expressions create anonymous functions. An ***anonymous function*** is a function that does not have a name. Anonymous functions are normally assigned to a variable so the function can be called, but an anonymous function can call itself when the anonymous function is declared. A ***self-invoking function*** is an anonymous function that invokes (calls) itself.

Figure 9.2.5: A self-invoking function.

```
// Anonymous function that calls itself
(function() {
   console.log("Hello, Anonymous!");
})();
```


Arrow functions

ECMAScript 6 defined a new way of declaring anonymous functions that is more compact than a function expression. An arrow function is defined with the => operator. Arrow functions are not yet supported by all browsers.

```
var findSum = (a, b) => a + b;
findSum(2, 3);   // returns 5

// Equivalent to
var findSum = function(a, b) {
   return a + b;
}
```

PARTICIPATION ACTIVITY 9.2.6: Function expressions.

1) The variable `z` is assigned 4.

```
var x = function(y) {
   return y * y;
}
var z = x(2);
```

- ○ True
- ○ False

2) The variable `z` is assigned 9.


```
var z = x(3);
var x = function(y) {
   return y * y;
}
```

- ○ True
- ○ False

3) The variable `z` is assigned 9.

```
var x = function(y) {
   return y * y;
}
var z = x;
```

- True
- False

4) The value 2 is output after the self-invoking function executes.

```
(function(x) {
  console.log(x);
})(2);
```

- True
- False

CHALLENGE ACTIVITY 9.2.1: Functions.

Start

Call the displayName function with parameter "Ron".

```
1 function displayName(userName) {
2    console.log("My name is " + userName + ".");
3 }
4
5 /* Your solution goes here */
6
```


1 2 3 4

Check Next

Exploring further:

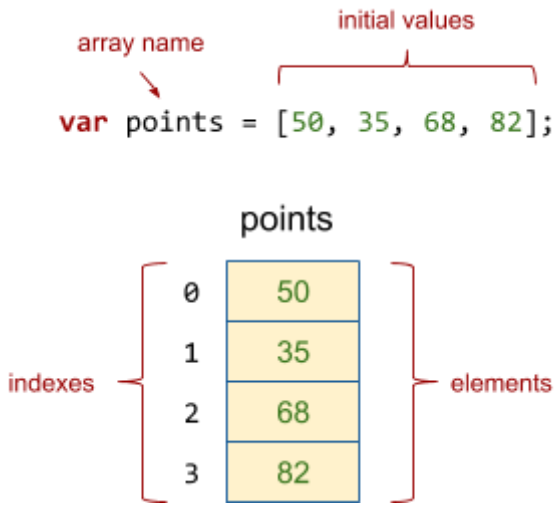
- Functions (MDN)

9.3 Arrays

Array introduction

An ***array*** is an ordered collection of values called ***elements***. Each array element is stored in a numeric location called an ***index***. Array elements may be of the same type or different types. Arrays increase in size as elements are added and decrease as elements are removed.

Figure 9.3.1: Array called points is initialized with 4 numbers.



PARTICIPATION ACTIVITY 9.3.1: Add, modify, and display array elements.

Animation captions:

1. Declare an empty array called "items".
2. Add three values to the items array at indexes 0, 1, and 2.

3. Modify the 3 elements in the items array.
4. Display the 3 elements in the items array.

PARTICIPATION ACTIVITY 9.3.2: Adding and displaying array elements.

1) Initialize `names` to an empty array.

```
var names = Answ;
```

Check Show answer

2) What is output to the console?

```
var names = [];
names[0] = "Sue";
names[1] = "Bob";
names[2] = "Jeff";
console.log(names[0] + names[1]);
```

Answer field

Check Show answer

3) What is output to the console?

```
var names = ["Sue", "Bob",
"Jeff"];
console.log(names[2]);
```

Answer field

Check Show answer

4) What is output to the console?

```
var names = ["Sue", "Bob",
"Jeff"];
console.log(names[3]);
```

Answer field

Check Show answer

5) What is output to the console?

```
var nums = [5, 2, 9, 3];
nums[0] += nums[1] + nums[2];
console.log(nums[0]);
```

Answer field

Check Show answer

6) What is output to the console?

```
function doSomething(n) {
   n[0]++;
}
var nums = [5, 2, 9, 3];
doSomething(nums);
console.log(nums[0]);
```


Answer field

Check Show answer

Adding and removing array elements

An array is an `Array` object. The `Array` object defines numerous methods for manipulating arrays. A ***method*** is a function that is attached to an object and operates on data stored in the object. Methods are called by prefacing the method with the object. Ex: `myArray.method();`.

Table 9.3.1: Array methods for adding and removing array elements.

Method	Description	Example

`.push(value)`	Adds a value to the end of the array	`var nums = [2, 4, 6];`
		`nums.push(8);`
		`// nums = [2, 4, 6, 8]`

`.pop()`	Removes the last array element and returns the element	`var nums = [2, 4, 6];` `var x = nums.pop(); // returns 6, nums = [2, 4]`

`.unshift(value)`	Adds a value to the beginning of the array	`var nums = [2, 4, 6];`
		`nums.unshift(0);`
		`// nums = [0, 2, 4, 6]`

`.shift()`	Removes the first array element and returns the	`var nums = [2, 4, 6];` `var x = nums.shift();`

Method	Description	Example

`.splice(startingIndex, totalElementsToDelete, valuesToAdd)`	Adds or removes elements from anywhere in the array and returns the deleted elements (if any)	`var nums = [2, 4, 6, 8, 10];`
		
		`// Deletes all elements from index 3 to the end`
		`nums.splice(3);`
		`// nums = [2, 4, 6]`
		
		`// Deletes 2 elements starting at index 0`
		`nums.splice(0, 2);  // nums = [6]`
		
		`// Adds 3, 5 starting at index 0`
		`nums.splice(0, 0, 3, 5);  // nums = [3, 5, 6]`
		
		`// Adds 7, 9, 11 starting at index 2`
		`nums.splice(2, 0, 7, 9, 11);`
		`// nums = [3, 5, 7, 9, 11, 6]`

PARTICIPATION ACTIVITY 9.3.3: Adding and removing array elements.

The six individuals in the `line` array are waiting in line. Write the JavaScript code to add or remove elements to/from the array to simulate the following events:

1. The person at the front of the line (index 0) leaves the line (`shift`).
2. The person at the end of the line cuts in front of the person at the front of the line (`pop` and `unshift`).
3. Two new people named "Poe" and "Snoke" cut into line behind the second person in line (`splice`).
4. The fifth person in line leaves the line (`splice`).
5. A new person named "Han" enters the back of the line (`push`).

Finally, display the contents of the `line` array to view the new line occupants. A correct solution will show: Leia, Finn, Poe, Snoke, Maz, Han.

```
// People waiting in line (Kylo is in front, Leia at the end)
var line = ["Kylo", "Finn", "Rey", "Maz", "Leia"];

// Show entire line
console.log(line);

```


Run JavaScript Reset code

Your console output

Looping through an array

The array property ***.length*** contains the number of elements in the array. The `.length` property is helpful for looping through an array using a `for` loop.

Figure 9.3.2: Looping through an array with a for loop.

```
var groceries = ["bread", "milk", "peanut butter"];

// Display all elements in groceries array
for (i = 0; i < groceries.length; i++) {
   console.log(i + " - " + groceries[i]);
}
```

```
0 - bread
1 - milk
2 - peanut butter
```


The array method ***.forEach()*** is also used for looping through an array. The `.forEach()` method takes a function as an argument. The function is called for each array element in order, passing the element and the element index to the function.

Figure 9.3.3: Looping through an array with a for loop.

```
var groceries = ["bread", "milk", "peanut butter"];

// Display all elements in groceries array
groceries.forEach(function(item, index) {
   console.log(index + " - " + item);
});
```

```
0 - bread
1 - milk
2 - peanut butter
```

PARTICIPATION ACTIVITY 9.3.4: Looping through an array.

1) What is `autos.length`?

```
var autos = ["Chevrolet", "Dodge",
"Ford", "Ram"];
```

- 0
- 3
- 4

2) What is output to the console?


```
var autos = ["Chevrolet", "Dodge",
"Ford", "Ram"];
for (i = 0; i < 2; i++) {
   console.log(autos[i]);
}
```

- Chevrolet, Dodge
- Chevrolet, Dodge, Ford
-

Chevrolet, Dodge, Ford, Ram

3) What is output to the console?

```
var autos = ["Chevrolet", "Dodge",
"Ford", "Ram"];
for (i = 0; i < autos.length; i++)
{
   if (i % 2 == 0) {
      console.log(autos[i]);
   }
}
```


- Chevrolet, Dodge, Ford, Ram
- Chevrolet, Ford
- Dodge, Ram

4) What is output to the console?

```
var autos = ["Chevrolet", "Dodge",
"Ford", "Ram"];
autos.forEach(function(item,
index) {
   if (index % 3 == 0) {
      console.log(item);
   }
});
```

- Chevrolet, Dodge, Ford, Ram
- Chevrolet, Ford
- Chevrolet, Ram

PARTICIPATION ACTIVITY 9.3.5: Practice looping.

Duke and North Carolina have a famous basketball rivalry dating back to 1920. The number of points each team has scored in head-to-head competition over the past five years is provided in the `dukeScores` and `ncScores` arrays. Ex: North Carolina won the most recent game 76-72 since `dukeScores[0]` is 72 and `ncScores[0]` is 76.

1. Write a `for` loop that examines the `dukeScores` and `ncScores` arrays and places "D" in the `winningTeam` array if Duke won or "N" if North Carolina won, for every game. Ex: `winningTeam[0]` should be "N" because North Carolina won 76-72, and `winningTeam[1]` should be "D" because Duke won 74-73.
2. Display the contents of the `winningTeam` array using `console.log(winningTeam);`.
3. Write a `forEach` loop that examines the `winningTeam` array and determines the longest streak of Duke wins. Display the longest streak to the console, which should be 4.

To determine the longest streak, use two variables initialized to 0: `streak` and `longestStreak`. Loop through the `winningTeam` array and increment `streak` every time a "D" appears in the array. When an "N" is encountered, set `longestStreak` to `streak` if `streak` is larger and reset `streak` back to 0. When the loop terminates, `longestStreak` will contain the longest streak.

```
1  var dukeScores  = [72, 74, 84, 92, 93, 66, 69, 73, 70, 85, 75, 67, 79];
2  var ncScores    = [76, 73, 77, 90, 81, 74, 53, 68, 88, 84, 58, 81, 73];
3  var winningTeam = [];
4
5  // Who won the last meeting?
6  if (dukeScores[0] > ncScores[0]) {
7     console.log("Duke won " + dukeScores[0] + "-" + ncScores[0] + ".");
8  }
9  else {
10    console.log("North Carolina won " + ncScores[0] + "-" + dukeScores[0] + "."
11 }
12
```


Run JavaScript Reset code

Your console output

for-in and for-of looping statements.

*The `for-in` and `for-of` statements are used to loop through arrays. The **for-in** statement iterates over the index of an array. The **for-of** statement iterates over*

iterable objects like an array.

```
var nums = [2, 4, 6];

for (var i in nums) {
   console.log(i);   // 0, 1, 2
}

for (var n of nums) {
   console.log(n);   // 2, 4, 6
}
```


*The **`for-in`** statement is not guaranteed to iterate over the array in the order of the index values, so **`for-in`** should not be used to loop through an array when index order is important. The **`for-of`** statement was added in ECMAScript 6, and not all browsers support **`for-of`** yet.*

Searching an array

The array methods ***.indexOf()*** and ***.lastIndexOf()*** search an array and return the index of the first found value or -1 if the value is not found. `.indexOf()` searches from the beginning of the array to the end. `.lastIndexOf()` searches from the end of the array to the beginning. Both functions take two arguments:

1. `searchValue` - The value to search for
2. `startingPosition` - Optional argument that indicates the index at which the search should begin (default is 0 for `.indexOf()` and `array.length - 1` for `.lastIndexOf()`)

Figure 9.3.4: Searching for array elements.

```
var scores = [80, 92, 75, 64, 88, 92];

s = scores.indexOf(92);          // 1
s = scores.indexOf(92, 2);       // 5
s = scores.indexOf(100);         // -1
s = scores.lastIndexOf(92);      // 5
s = scores.lastIndexOf(92, 4);   // 1
s = scores.lastIndexOf(50);      // -1
```


PARTICIPATION ACTIVITY 9.3.6: Searching an array.

Refer to the `artists` array.

```
var artists = ["Raphael", "Titian", "Masaccio", "Botticelli", "Titian"];
```

1) `artists.indexOf("Botticelli")` returns 3.

- True
- False

2) `artists.lastIndexOf("Titian")` returns 1.

- True
- False

3) `artists.indexOf("Michaelangelo")` returns NaN.

- True
- False

PARTICIPATION ACTIVITY 9.3.7: Practice searching an array.

The `validCredentials()` function contains two parallel arrays of usernames and passwords. Modify `validCredentials()` to use the `.indexOf()` method to search the `usernames` array for the given `enteredUsername`. If the username is found, the same location in the `passwords` array should contain the `enteredPassword`. Return `true` if the passwords are equal, `false` otherwise. `validCredentials()` should also return `false` if the given username was not found.

```
1  // Return true if the given username and password are in the database,
2  // false otherwise.
3  function validCredentials(enteredUsername, enteredPassword) {
4
5     // Database of usernames and passwords
6     var usernames = ["smith",  "tron",      "ace",      "ladyj",    "anon"];
7     var passwords = ["qwerty", "EndOfLine", "year1942", "ladyj123", "PASSWORD"]
8
9     // Search the usernames array for enteredUsername
10
11    // Only return true if the enteredUsername is in username, and the
12    // same location in passwords is enteredPassword
13    return true;
14 }
15
16
17 console.log("Login for ladyj: " + validCredentials("ladyj", "ladyj123"));  //
18 console.log("Login for ace: " + validCredentials("ace", "wrong"));  // false
19 console.log("Login for jake: " + validCredentials("jake", "???"));  // false
```

Run JavaScript Reset code

Your console output

Sorting an array

The array method ***.sort()*** sorts an array in ascending (increasing) order. `.sort()`'s default behavior is to sort each element as a string using the string's Unicode values. Sorting by Unicode values may yield unsatisfactory results for arrays that store numbers. Ex: 10 is sorted before 2 because "10" is < "2" when comparing the Unicode values of "1" to "2".

The `.sort()` method can sort elements in other ways by passing a comparison function to `.sort()`. The comparison function returns a number that helps `.sort()` determine the sorting order of the array's elements:

- Returns a value < 0 if the first argument should appear before the second argument.
- Returns a value > 0 if the first argument should appear after the second argument.
- Returns 0 if the order of the first and second arguments does not matter.

Figure 9.3.5: Sorting an array of numbers.


```
var numbers = [200, 30, 1000, 4];

// Sort based on Unicode values: [1000, 200, 30, 4]
numbers.sort();

// Sort numbers in ascending order: [4, 30, 200, 1000]
numbers.sort(function(a, b) {
   return a - b;
});
```

PARTICIPATION ACTIVITY 9.3.8: Sorting arrays.

1) What is output to the console?

```
var names = ["Sue", "Bob",
"Jeff"];
names.sort();
console.log(names[0]);
```


Answer field

Check Show answer

2) What is output to the console?

```
var names = ["Sue", "Bob",
"Jeff"];
names.sort(function(a, b) {
   if (a > b) {
      return -1;
   }
   else if (a < b) {
      return 1;
   }
   return 0;
});
console.log(names[0]);
```

Answer field

Check Show answer

3) What is output to the console?

```
var totals = [99, 4, 250, 38];
totals.sort();
console.log(totals[0]);
```

Answer field

Check Show answer

4) What is output to the console?

```
var totals = [99, 4, 250, 38];
totals.sort(function(a, b) {
   return b - a;
});
console.log(totals[0]);
```

Answer field

Check Show answer

CHALLENGE ACTIVITY 9.3.1: Arrays.

Start

Display elements at indices 0 and 4 in the array numberList separated by a space.

```
1 var numberList = [1, 6, 41, 8, 24, 4]; // Tests may use different array values
2
3 /* Your solution goes here */
4
```

1 2 3 4

Check Next

Exploring further:

- Array object (MDN)

9.4 Objects

Objects and properties

An ***object*** is an unordered collection of properties. An object ***property*** is a name-value pair, where the name is a string and the value is any data type. Objects are often defined with an object literal. An ***object literal*** (also called an ***object initializer***) is a comma-separated list of property name and value pairs.

PARTICIPATION ACTIVITY 9.4.1: Creating an object with an object literal.

Animation captions:

1. book is assigned an empty object literal.
2. book is assigned an object literal with three properties: title, published, keywords.
3. Display the title and first keyword of the book object.
4. book is assigned an object literal with an embedded object literal that is assigned to the author property.
5. Display the last name of the book's author.

PARTICIPATION ACTIVITY 9.4.2: Accessing object properties.

Use the object below to answer the questions.

```
var book = {
   title: "Hatching Twitter",
   published: 2013,
   keywords: ["origins", "betrayal", "social media"],
   author: {
      firstName: "Nick",
      lastName: "Bilton"
   }
};
```

1) Which statement changes the published year to 2014?

- book.Published = 2014;
- book.published = 2014;
- book.published : 2014;

2) Which statement adds a new property called "isbn" with the value "1591846013"?

- book.isbn = "1591846013";
- isbn = "1591846013";
-

book.isbn("1591846013");

3) What does the following statement do?

```
book.author["firstName"] = "Jack";
```

- Produces a syntax error.
- Changes `author` into an array.
- Replaces "Nick" with "Jack".

4) What is missing from the code below to remove "social media" from the book's keywords?

```
_______.pop();
```

- keywords
- book.keywords[2]
- book.keywords

5) What is output to the console?

```
var test = book;
test.title = "Catching Glitter";
console.log(book.title);
```

- Catching Glitter
- Hatching Twitter
- undefined

Methods

An object property may be assigned an anonymous function to create a method. Methods may access the object's properties using the keyword `this`, followed by a period, before the property name. Ex: `this.someProperty`.

Figure 9.4.1: Defining a method in an object literal.

```
var book = {
   title: "Quiet",
   author: {
      firstName: "Susan",
      lastName: "Cain"
   },

   // Define a method
   getAuthorName: function() {
      return this.author.firstName + " " + this.author.lastName;
   }
};

// Call a method that returns "Susan Cain"
var name = book.getAuthorName();
```


Figure 9.4.2: Defining a method for an existing object.

```
var book = {
   title: "Quiet",
   author: {
      firstName: "Susan",
      lastName: "Cain"
   }
};

// Define a method
book.getAuthorName = function() {
   return this.author.firstName + " " + this.author.lastName;
};

// Call a method that returns "Susan Cain"
var name = book.getAuthorName();
```

PARTICIPATION ACTIVITY 9.4.3: Object methods.

Refer to the above figures.

1) A method may be defined inside or outside an object literal.

   - True
   - False

2) The method below outputs "I'm

reading 'Quiet'.".

```
book.read = function() {
   console.log("I'm reading '" +
title + "'.");
};
```

- ○ True
- ○ False

3) The method below creates a new object property.

```
book.assignMiddleInitial =
function(middleInitial) {
   this.author.middleInitial =
middleInitial;
};

book.assignMiddleInitial("H");
```

- ○ True
- ○ False

PARTICIPATION ACTIVITY 9.4.4: Practice creating objects and methods.

Create an object called `game` that represents a competition between two opponents or teams. Add the following properties to `game`, and assign any value to each property:

1. `winner` - An object with properties `name` and `score`
2. `loser` - An object with properties `name` and `score`

Add the following methods to `game`:

1. `getMarginOfVictory()` - Returns the difference between the winner's score and the loser's score
2. `showSummary()` - Outputs to the console the winner's name and score and the loser's name and score

Call the two methods to verify the methods work correctly. Example output:


```
Broncos: 24
Panthers: 10
Margin of victory = 14
```

JavaScript | CSS

```
1 // Declare a game object and call the game object's methods
2
```


Run JavaScript Reset code

Your console output

Accessor properties

An object property may need to be computed when retrieved, or setting a property may require executing some code to perform data validation. The `get` and `set` keywords define getters and setters for a property. A ***getter*** is a function that is called when an object's property is retrieved. Syntax to define a getter: `get property() { return someValue; }`. A ***setter*** is a function that is called when an object's property is set to a value. Syntax to define a setter: `set property(value) { ... }`. An ***accessor property*** is an object property that has a getter or a setter or both.

Figure 9.4.3: Defining an accessor property called 'area'

Figure 9.4.3: Defining an accessor property called "area".

```
var rectangle = {
   width: 5,
   height: 8,
   get area() {
      return this.width * this.height;
   },
   set area(value) {
      // Set width and height to the square root of the value
      this.width = Math.sqrt(value);
      this.height = this.width;
   }
};

var area = rectangle.area;     // Calling getter returns 40
rectangle.area = 100;          // Calling setter sets width and height to 10
console.log(rectangle.width);  // 10
```


PARTICIPATION ACTIVITY 9.4.5: Accessor properties.

Refer to the `game` object.

```
var game = {
   firstOpponent: "Serena Williams",
   firstOpponentScore: 2,
   secondOpponent: "Garbine Muguruza",
   secondOpponentScore: 0,
   get winner() {
      if (this.firstOpponentScore > this.secondOpponentScore) {
         return this.firstOpponent;
      }
      else if (this.secondOpponentScore > this.firstOpponentScore) {
         return this.secondOpponent;
      }
      else {
         return "Tie";
      }
   }
};
```

1) The code below outputs "Serena Williams".

```
console.log(game.winner());
```


- ○ True
- ○ False

2) The code belows outputs "Maria Sharapova" .

```
game.winner = "Maria Sharapova";
console.log(game.winner);
```

- ○ True
- ○ False

3) The `matchDate` setter below sets the `date` property to the given `value`.


```
var game = {
   ...
   date: "",
   set matchDate(value) {
      date = value;
   },
   ...
};
```

- ○ True
- ○ False

4) What sets the game's match date to June 9, 2016?

```
var game = {
   ...
   date: "",
   set matchDate(value) {
      this.date = value;
   },
   ...
};

// Wimbledon 2016 women's championship
var date = new Date(2016, 5, 9);
```

- ○ game.matchDate = date;
- ○ game.matchDate(date);

PARTICIPATION ACTIVITY 9.4.6: Practice creating accessor properties.

The `musicQueue` object contains a `songs` property listing all the songs in the music queue. Add an accessor property called "next" with the following accessors:

- getter - Returns the song in the `songs` array at index `nextSong` and increments `nextSong` by one so the next song in the queue will be retrieved the next time the getter is accessed. If `nextSong` is beyond the boundaries of the `songs` array, `nextSong` should be assigned 0.

- setter - Sets `nextSong` to the given value. If the value is outside the `songs` array's bounds, `nextSong` should be assigned 0.

If the `next` property is implemented correctly, the for loop under the `musicQueue` will display each of the 3 songs 3 times. The code under the for loop tests the setter and should display the song in comments.

```
var musicQueue = {
   songs: ["Party Rock Anthem", "I Gotta Feeling", "Macarena"],
   nextSong: 0

   // Add getter and setter for next property
};

// Run through the queue three times
for (var c = 0; c < musicQueue.songs.length * 3; c++) {
   console.log("Now playing: " + musicQueue.next);
}

// Test the next setter
musicQueue.next = 2;
console.log(musicQueue.next);   // Macarena
musicQueue.next = 3;
console.log(musicQueue.next);   // Party Rock Anthem
musicQueue.next = -1;
console.log(musicQueue.next);   // Party Rock Anthem
```


Run JavaScript Reset code

Your console output

Objects as associative arrays

An ***associative array*** or ***map*** is a data structure that maps keys to values. JavaScript objects can be used as associative arrays, in which the key is the object property and the value is the

property's value.

Figure 9.4.4: Defining an associative array of state capitals.

```
var capitals = {
   AR: "Little Rock",
   CO: "Denver",
   NM: "Sante Fe"
};

console.log(capitals["AR"]);   // Little Rock
console.log(capitals["CO"]);   // Denver
console.log(capitals["NM"]);   // Sante Fe
```


The `for-in` is ideal for looping through an associative array. The ***for-in loop*** iterates over an object's properties in arbitrary order.

Construct 9.4.1: for-in loop.

```
for (variable in object) {
   body
}
```

Figure 9.4.5: Looping through an associative array with for-in.

```
var capitals = {
   AR: "Little Rock",
   CO: "Denver",
   NM: "Sante Fe"
};

for (var state in capitals) {
   console.log("The capital of " + state + " is " + capitals[state] + ".");
}
```

```
The capital of AR is Little Rock.
The capital of CO is Denver.
The capital of NM is Sante Fe.
```


The `Object.keys()` method returns an array of an object's property names, and is often used to determine the number of elements in an associative array.

Figure 9.4.6: Using Object.keys() to determine an associative array's size.

```
var capitals = {
   AR: "Little Rock",
   CO: "Denver",
   NM: "Sante Fe"
};

var states = Object.keys(capitals);
console.log(states);          // AR,CO,NM
console.log(states.length);   // 3
```


PARTICIPATION ACTIVITY 9.4.7: Associative arrays.

Refer to the associative array below.

```
var contacts = {
   Ann: {
      phone: "555-4321",
      email: "ann@gmail.com"
   },
   Dave: {
      phone: "533-9988",
      email: "dave@yahoo.com"
   },
   Zack: {
      phone: "511-6758",
      email: "zack@msn.com"
   }
};
```

1) What outputs Dave's email address?

```
console.log(_____);
```

- ["Dave"].email
- contacts.email
- contacts["Dave"].email

2) What assigns a Twitter username to Ann?

```
_____ = "@annLuvsCats";
```

- contacts["Ann"].twitter
- contacts["ann"].twitter
- contacts["Ann"].email

3) What adds John to the `contacts` associative array?

```
_______ = { phone: "111-2222",
email: "john@work.org" };
```

- `contacts["John"].email`
- `contacts`
- `contacts["John"]`

4) Which expression loops through the `contacts` array to output all names and phone numbers?

```
for (_______) {
   console.log(name + ": " +
contacts[name].phone);
}
```

- `contacts`
- `name in contacts`
- `contacts in name`

PARTICIPATION ACTIVITY 9.4.8: Practice with associative arrays.

Create an associative array called `courses` that stores a university department's course number as the key and an object as the value. The object has 3 properties: `title`, `description`, `creditHours`. Example courses:

- 170 - Introduction to Programming, Develop algorithms for computers, 5.
- 250 - Web Development, Build web applications, 3.
- 310 - Operating Systems, Process management and memory management, 3.
- 430 - Artificial Intelligence, Simulate human thinking, 2.

Output the total number of courses in the `courses` object using `Object.keys()`. Then, write a `for-in` loop that displays the course number and title for only those courses that are 3 credit hours.

```
1 // Define the courses associative array
2
3 // Display all the values in the courses associative array
4
```


Run JavaScript Reset code

Your console output

Deleting elements in an associative array

The `delete` operator removes keys/properties from an associative array or object. The `in` operator returns `true` if an object contains the given property and returns `false` otherwise.

Figure 9.4.7: Using the "delete" and "in" operators.

```
// Define an empty associative array
var capitals = {};

// Add an element
capitals["NM"] = "Santa Fe";

// Evaluates true
if ("NM" in capitals) {
   console.log("NM exists");
}

// Remove the NM/Santa Fe pair
delete capitals["NM"];

// Evaluates false
if ("NM" in capitals) {
   console.log("NM exists");
}

// Outputs undefined
console.log(capitals["NM"]);
```


PARTICIPATION ACTIVITY 9.4.9: in and delete operators.

Refer to the associative array below.

```
var students = {
   123: { name: "Tiara",  gpa: 3.3 },
   444: { name: "Lee",    gpa: 2.0 },
   987: { name: "Braden", gpa: 3.1 }
};
```

1) Remove Lee from `students`.

```
delete ________;
```

Answer field

Check Show answer

2) What number is output to the console?

```
delete students["Braden"];
console.log(Object.keys(students).length);
```

Answer field

Check Show answer

3) What is missing to check if student ID 888 is in `students`?

```
if (_________) {
   console.log("Hello, " +
students[888].name);
}
```

Answer field

Check Show answer

4) What is output to the console?

```
for (var id in students) {
   delete students[id];
}
if (123 in students) {
   console.log("yes");
}
else {
   console.log("no");
}
```

Answer field

Check Show answer

Map object

ECMAScript 6 defines a `Map` object that stores key/value pairs. The `Map` object provides a few advantages over an `Object` for storing maps or associative arrays. However, not all browsers support the `Map` object yet.

```
var capitals = new Map();
capitals.set("AR", "Little Rock");
capitals.set("CO", "Denver");
capitals.set("NM", "Santa Fe");

capitals.size;       // 3
capitals.get("CO");  // Denver
```


CHALLENGE ACTIVITY 9.4.1: Objects.

Start

Display the movie's composer.

```
var movie = { // Code will be tested with a different movie
   name: "Interstellar",
   director: "Christopher Nolan",
   composer: "Hans Zimmer",
   cast: {
      "Matthew McConaughey": "Cooper",
      "Anne Hathaway": "Brand",
      "Jessica Chastain": "Murph",
      "Matt Damon": "Mann",
      "Mackenzie Foy": "Young Murph"
   }
};

/* Your solution goes here */
```


1 2 3 4 5

Check Next

Exploring further:

- Working with objects (MDN)
- Map object (MDN)