**Loops**

One of a computer's greatest abilities is to repeat a task multiple times. Loops let us tell the computer to loop over a block of code so that we don't have to write out the same process over and over.

Loops are especially useful when we have an array where we'd like to do something to each of its items, like logging each item to the console.

There are two kinds of loops we will learn in this lesson:

1. for loops, which let us loop a block of code a known amount of times.
2. while loops, which let us loop a block of code an unknown amount of times.

**for Loops**

Instead of writing out the same code over and over, let’s make the computer loop through our array for us. We can do this with for loops.

The syntax looks like this:

let animals = ["Grizzly Bear", "Sloth", "Sea Lion"]; for (let animalIndex = 0; animalIndex < animals.length; animalIndex++) { console.log(animals[animalIndex]); }

The output would be the following:

Grizzly Bear Sloth Sea Lion

Since this syntax is a little complicated, let's break it into four parts:

1. Within the for loop's parentheses, the *start condition* is let animalIndex = 0, which means the loop will start counting at 0. animalIndex is called an *iterator variable*and it is a good practice to give this variable a descriptive name.
2. The *stop condition* is animalIndex < animals.length, which means the loop will run as long as animalIndex is less than the length of the animals array. When animalIndex is equal to the length of the animals array, the loop will stop executing.
3. The *iterator* is animalIndex++. This means that after each loop, animalIndex will increase by 1.
4. Finally, the code block is inside of the { and } curly braces. The block will execute each loop until the program reaches the stop condition.

The secret to loops is that animalIndex, the variable we created inside the for loop's parentheses, is equal to a number. To be more clear, the first loop, animalIndex will equal 0, the second loop, animalIndex will equal 1, and the third loop, animalIndex will equal 2.

Loops make it possible to write animals[0], animals[1], animals[2] programmatically instead of by hand. We can write a for loop and replace the hard-coded number with the variable animalIndex, like this: animals[animalIndex].

|  |
| --- |
| let vacationSpots = ['Mozambique', 'Thailand', 'Bolivia'];  for (let vacationSpotIndex = 0;vacationSpotIndex<vacationSpots.length;vacationSpotIndex++ ){  console.log('I would love to visit ' + vacationSpots[vacationSpotIndex]);  } |
| I would love to visit Mozambique  I would love to visit Thailand  I would love to visit Bolivia |

# for Loops, Backwards

If we can make a for loop run forwards through an array, can we make it run backward through one? Of course!

We can make our loop run backward by modifying the start, stop, and iterator conditions.

To do this, we'll need to edit the code between the for loop's parentheses:

1. The start condition should set vacationSpotIndex to the length of the array.
2. The loop should stop running when vacationSpotIndex is less than 0.
3. The iterator should subtract 1 each time, which is the purpose of vacationSpotIndex--.

# Nested for Loops

Let's say that you and a friend would like to go on vacation together. You've both made arrays of your favorite places and you want to compare to see if any of them match. This is a job for loops!

The big idea is that we can run a for loop inside another for loop to compare the items in two arrays.

Every time the outer for loop runs once, the inner for loop will run completely.

These are called nested for loops and we can use them to check to see if any of your vacation spots match your friend's spots.

See the example below for proper formatting of nested for loops.

for (let i = 0; i < myArray.length; i++) { for (let j = 0; j < yourArray.length; j++) { //Code To Run } }

Note that in the example above, we used i and j as the iterator variables to make the structure of the code easier to see, but it is a better practice to use descriptive variable names.

**Instructions**

**1.**

We are going to write this program from scratch. Start out by writing a variable named myPlaces and set it equal to an array with three places you'd like to visit.

**2.**

Now, make another variable named friendPlaces and set it equal to an array with three places a friend might like to visit.

Make sure that at least one of the places is the same as in your myPlaces array.

**3.**

Write a for loop that iterates through each item in myPlaces and logs out each place.

Use myPlacesIndex as the iterator variable.

Hint

for loops follow this pattern:

for (let i = 0; i < myArray.length; i++) { console.log(myArray[i]); }

**4.**

You logged all of your places!

Inside of the existing for loop's block, below the console.log() statement, write another for loop that loops over friendPlaces. This time, rather than using the myPlacesIndex as the variable name, use friendPlacesIndex. Doing so will prevent us from overwriting any variables. Log each of your friend's places to the console.

Hint

for (let i = 0; i < myArray.length; i++) { console.log(myArray[i]); for (let j = 0; j < yourArray.length; j++) { console.log(yourArray[j]); } }

**5.**

Look what printed to the console. Your first place printed, then all three of your friend's. Then your second place, then your friend's places again. Finally, your third place printed and all three of your friend's places printed a final time.

This is because the inner for loop runs completely every time the outer forloop runs once.

The purpose of the program is to see what you and your friend have in common. Let's utilize the ===comparison with an if statement.

Delete the two console.log()statements you wrote in the previous steps.

Inside the second for loop's block, write an if statement that checks if myPlaces[myPlacesIndex] is equal to friendPlaces[friendPlacesIndex]. If it is, log to the console the place you have in common.

Hint

if (array[i] === otherArray[j]) { console.log('Match: ' + array[i]); }

|  |
| --- |
| let myPlaces = ['NewYork', 'Taiwan', 'Japen'];  let friendPlaces = ['Taiwan', 'China', 'Japen'];  for (let myPlacesIndex=0; myPlacesIndex<myPlaces.length;myPlacesIndex++){  console.log(myPlaces[myPlacesIndex]);  for (let friendPlacesIndex=0; friendPlacesIndex<friendPlaces.length;friendPlacesIndex++){  if (myPlaces[myPlacesIndex]===friendPlaces[friendPlacesIndex]){  console.log('Match: ' + friendPlaces[friendPlacesIndex]);  }  }  } |

**while Loops**

Awesome job! for loops are great, but they have a limitation: you have to know how many times you want the loop to run. What if you want a loop to execute an unknown or variable number of times instead?

For example, if we have a deck of cards and we want to flip cards (loop a card flipping function) until we get a Spade, how could we write that in JavaScript?

That's the purpose of the while loop. It looks similar to a for loop. See the example below.

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

1. The loop begins with the keyword while.
2. Inside the parentheses, we write a condition. As long as the condition evaluates to true, the block of code will loop.
3. Inside the code block, we can write any code we'd like to loop.

**Instructions**

**1.**

Below the array cards, create a variable named currentCard and set it equal to 'Spade'.

This variable will hold the name of the card we just flipped. We are using 'Spade' as the first card.

**2.**

Create a while loop. The condition should check if the currentCard is NOT a 'Spade'.

**3.**

Inside the while loop, log the value of currentCard to the console.

Inside the while loop, below the console.log() statement you just wrote, add this code:

currentCard = cards[Math.floor(Math.random() \* 4)];

This code will generate a random number between 0 and 3, the range of indices of the cards array, and reassign currentCard to a new card from that array. Because the while loop only runs if the card is NOT a Spade, the value of currentCard will only be logged to the console if it is not 'Spade'.

**4.**

Outside the while loop, on the last line of the program, use console.log() to log that the program found a spade. You can use the string "found a spade" inside console.log().

Change the currentCard that you set to 'Spade' in the first step to 'Heart' so that the program will run.

Run the code a few times to see the output changing. You can see the whileloop guessing a card, then seeing if it is a Spade, over and over, until it finds one.

Hint

Did you remember to change the currentCardyou set at the beginning of this exercise to 'Heart' before running your code?

|  |
| --- |
| let cards = ['Diamond', 'Spade', 'Heart', 'Club'];  let currentCard = 'Spade';  while(currentCard != 'Spade'){  console.log(currentCard);  currentCard = cards[Math.floor(Math.random() \* 4)];  }  console.log('Found a spade!'); |
| Found a spade! |

|  |
| --- |
| let cards = ['Diamond', 'Spade', 'Heart', 'Club'];  let currentCard = 'Heart';  while(currentCard != 'Spade'){  console.log(currentCard);  currentCard = cards[Math.floor(Math.random() \* 4)];  }  console.log('Found a spade!'); |
| Heart  Club  Club  Heart  Found a spade! |

**Infinite Loops**

All loops should execute a finite number of times, infinite loops don't reach a stopping point. An infinite loop means there was an error in the logic of the loop because the stopping condition is never reached.

This type of error is harder to spot because your syntax is correct and your code will execute. When an infinite loop occurs, it can freeze your computer, making your computer unresponsive to your commands.

When you initially work with loops, you may accidentally create infinite loops. It is important to be aware of infinite loops so you can avoid them.

Both for loops and while loops need explicit instructions on when to terminate. Infinite loops are more common in while loops because they don't have an iterator built into their syntax.

When writing a while loop, be sure to write code that will guarantee the condition will eventually be met.

A common infinite loop occurs when the condition of the while statement is set to true. Below is an example of code that will run forever.

// Initiate an infinite loop while (true) { // execute code forever }

for loops require a start condition, a stop condition, and an iterator. The iterator should bring the loop from the start condition to the stop condition.

for (let i = 0; i < array.length; i--) { //some code }

1. The loop begins with i = 0.
2. After one iteration through the loop, i is equal to -1. This is because i begins at 0and 1 is subtracted from i each loop.

Do you see the problem? i is decreasing each time. It will never equal the length of the array. This code will execute forever.

When we change the iterator to i++, as in the example below, i will eventually equal the length of the array. We have eliminated the infinite loop!

for (let i = 0; i < array.length; i++) { //some code }

To avoid infinite loops, make sure to properly initialize the counter and make sure the terminating condition is eventually met with the proper updates to the counter variable.

**Instructions**

In **main.js**, notice the while loop that has been written. You can run the code to see the loop in action.

The part of this code that ensures the loop is not infinite is:

if (counter ===37){ break; }

This code will terminate the while loop, with the break keyword, when the condition counter === 37 is met. Without this code the while loop would run infinitely because there is no condition stated to "break" the loop.

|  |
| --- |
| let flag = true;  let counter = 0;  while(flag === true){  console.log(counter);  counter+=1;  if (counter === 37){  break;  }  } |

**Review: Loops**

Great job! In this lesson, we learned how to write less repetitive code with loops.

* for loops allow us to repeat a block of code a known amount of times.
* We can use a for loop inside another forloop to compare two lists.
* while loops are for looping over a code block an unknown amount of times.
* Infinite loops occur when stop conditions are never met.

|  |
| --- |
| * let groceryList = ['orange juice', 'bananas', 'coffee beans', 'brown rice', 'pasta', 'coconut oil', 'plantains']; * groceryList.shift(); * groceryList.unshift('popcorn'); * console.log(groceryList); * console.log(groceryList.slice(1,4)); * console.log(groceryList); |