JavaScript Essentials

Loops

Table of Contents

Lesson Objectives

- Understand the importance of loops in Programming
- Able to use loops in JavaScript to complete repetitive tasks
- Understand the difference of for, while and do…while loop
- Know when to use for loop and when to use while and do…while loop

Section 1

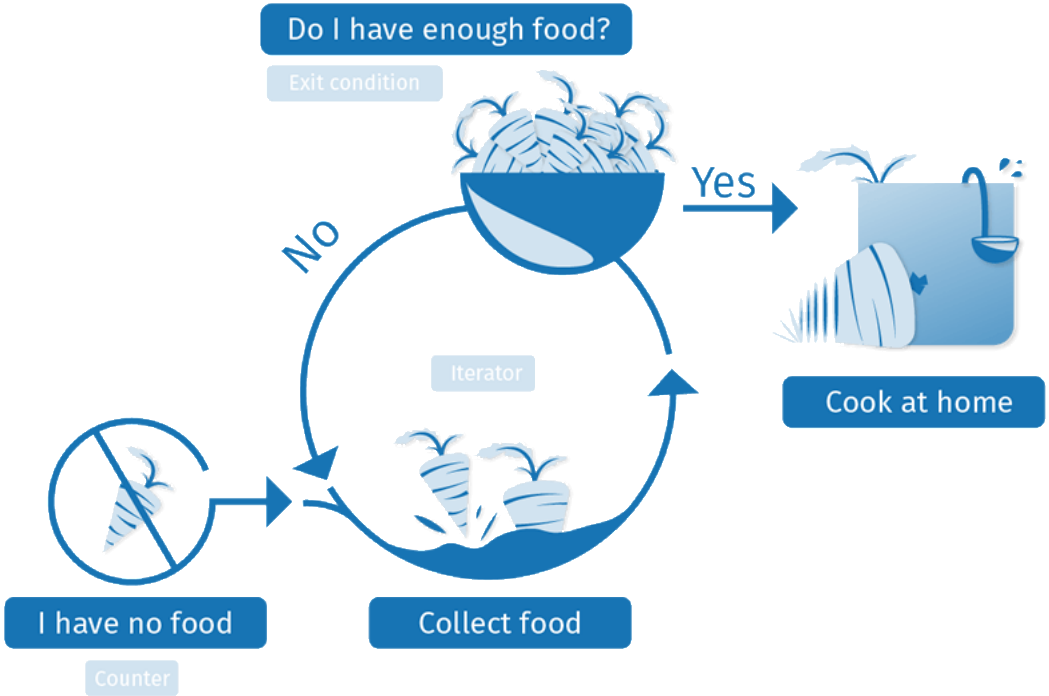
Overview

Overview

- Programming languages are very useful for rapidly completing repetitive tasks, for example: compute sum, product of an array
- Programming loops are all to do with doing the same thing over and over again — which is termed **iteration** in programming speak.

Overview

- Let's consider the case of a farmer that is making sure he has enough food to feed his family for the week. He might use the following loop to achieve this:



Overview

- A loop usually has one or more of the following features:
  1. A **counter**
  2. An **exit condition**
  3. An **iterator**

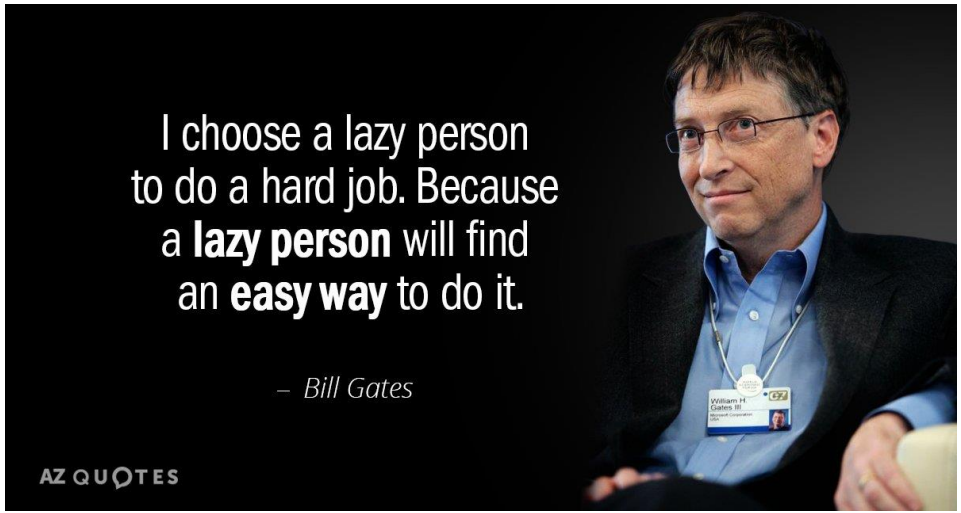
Overview

In pseudocode, this would look something like the following:

```
loop(food = 0; foodNeeded = 10) {
  if (food >= foodNeeded) {
    exit loop;
    // We have enough food; let's go home
  } else {
    food += 2; // Spend an hour collecting 2 more food
    // loop will then run again
  }
}
```

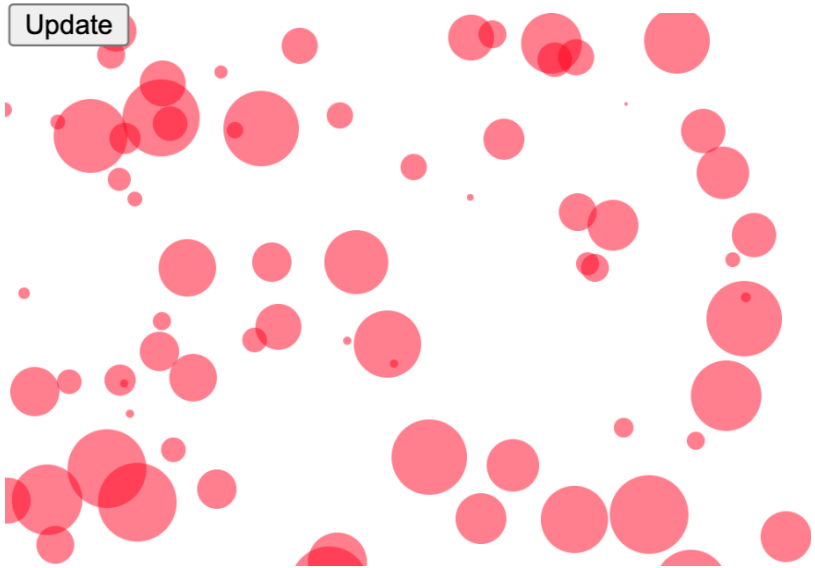
Overview – Why loop?

- **loops are all to do with doing the same thing over and over again**, which is great for **rapidly completing repetitive tasks**.
- Loops are what make you into a Programmer
- Remember what Bill Gates said?



Overview – Why loop?

Demo 1: Let's say we wanted to draw 100 random circles on a <canvas> element (press the ***Update*** button to run the example again and again to see different random sets)



Overview – Why loop?

```
// with loops
for (let i = 0; i < 100; i++) {
  ctx.beginPath();
  ctx.fillStyle = 'rgba(255,0,0,0.5)';
  ctx.arc(random(WIDTH), random(HEIGHT), random(50), 0, 2 * Math.PI);
  ctx.fill();
}
```

Overview – Why loop?

```
// without loops
ctx.beginPath();
ctx.fillStyle = 'rgba(255,0,0,0.5)';
ctx.arc(random(WIDTH), random(HEIGHT), random(50), 0, 2 * Math.PI);
ctx.fill();
```

```
ctx.beginPath();
ctx.fillStyle = 'rgba(255,0,0,0.5)';
ctx.arc(random(WIDTH), random(HEIGHT), random(50), 0, 2 * Math.PI);
ctx.fill();
```

```
ctx.beginPath();
ctx.fillStyle = 'rgba(255,0,0,0.5)';
ctx.arc(random(WIDTH), random(HEIGHT), random(50), 0, 2 * Math.PI);
ctx.fill();
```

Same task

Overview – Summary

- Loops are all to do with doing the same thing over and over again (which is termed **iteration** in programming speak)
- Loops are essentials for rapidly completing repetitive tasks
- Every time you do a same tasks over and over think about loops
- Remember the mind of Programmer: **Don’t repeat – be lazy**

Section 2

Famous for loop

Famous for loop

- Let's start exploring some specific loop constructs. The first, which you'll use most of the time, is the for loop — this has the following syntax:

```
for (initializer; exit-condition; final-expression) {
  // code to run
}
```

Famous for loop

- Demo 2: print out cat names using **for** loop

```
My cats are called Bill, Jeff, Pete, Biggles, Jasmin,
```

Famous for loop

- One small problem we are left with is that the final output sentence isn't very well-formed
- Ideally, we want to change the concatenation on the final loop iteration so that we haven't got a comma on the end of the sentence

```
// step 2:
for (var i = 0; i < cats.length; i++) {
  if (i === cats.length - 1) {
    // last element;
    info += ' and ' + cats[i] + '.';
  } else {
    info += cats[i] + ', ';
  }
}
```

Famous for loop - Important

- **Attention: m**ake sure that the initializer is incremented or, depending on the case, decremented, so that it eventually reaches the exit condition.
- If not, the loop will go on forever, and either the browser will force it to stop, or it will crash. This is called an **infinite loop**

Loop backward from last to first element

Infinite loops

```
// infinite loops
for (var i = cats.length; i ⩾ 0; i++) {
  // code
}
```

But index never reaches exit condition

Famous for loop - Overview

- Syntax for for loop is: for (initializer; exit-condition; final-expression) { // code to run }
- For loop is commonly used when you know exactly the number of iteration for example loop every item of an array (n iteration where n is the length of the array)
- Remember Computer starts a 0 so last item when you loop an array is n – 1 (where n is the length of the array)
- Take care of the loop condition or you will get an **Infinite loops** (which is very bad)

Section 3

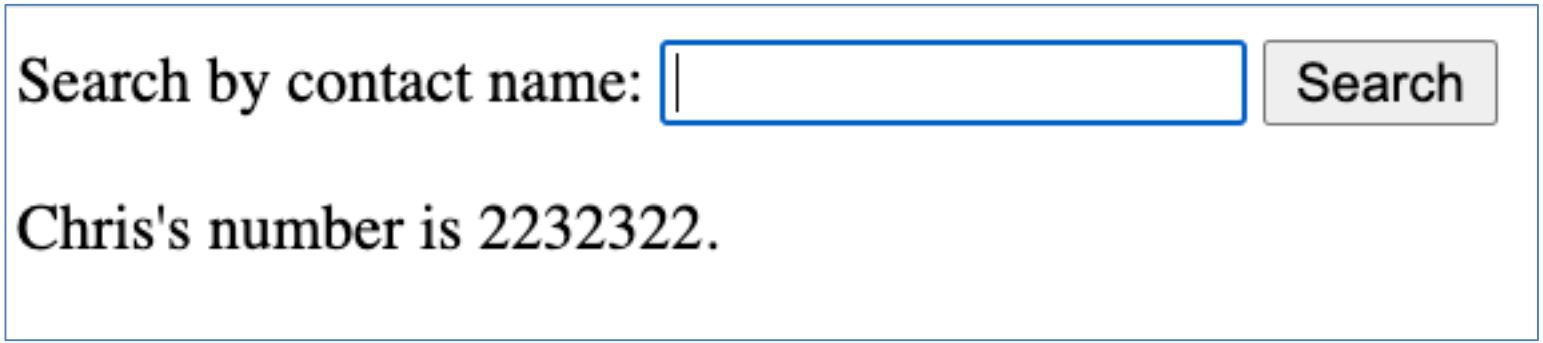
Break the loops

Break the loops

- If you want to exit a loop before all the iterations have been completed, you can use the break statement.
- We already met this in the previous article when we looked at switch statements
- It's the same with loops — a break statement will immediately exit the loop and make the browser move on to any code that follows it.

Break the loops - Example

- Demo 3: Contact search



Break the loops – Practice 1

- Practice 1: Use break to loop the 1st half of an array ?

Break the loops - Overview

- A break statement will immediately exit the loop and make the browser move on to any code that follows it.
- Use break when you want to loop a partition of the iterations for example: loop first half of an array
- Use **break** outside of loops does not make sense

Section 4

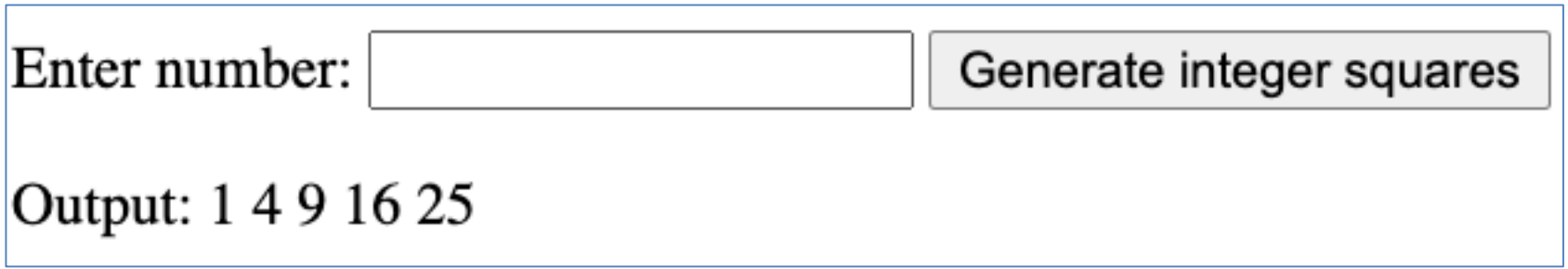
Skip iterations with continue

Skip iterations with continue

- The continue statement works in a similar manner to break, but instead of **breaking** out of the loop entirely, it **skips** current iteration and to the next iteration of the loop.

Skip iterations with continue

- Demo 4: print all square root of a given number



Skip iterations with continue - Summary

- To skips to the next iteration of the loop use **continue**
- continue is commonly used with conditionals statement such as if…else
- Use **continue** outside of loop makes no sense

Section 5

while and do…while

while and do…while

- for is not the only type of loop available in JavaScript.
- There are actually many others and, while you don't need to understand all of these now, it is worth having a look at the structure of a couple of others so that you can recognize the same features at work in a slightly different way.

while and do…while

- First, let's have a look at the while loop. This loop's syntax looks like so:

```
initializer
while (exit-condition) {
  // code to run

  final-expression
}
```

while and do…while

- This works in a very similar way to the **for** loop
- Except that:
  1. the **initializer** variable is set before the loop, and the **final-expression** is included inside the loop after the code to run
  2. The **exit-condition** is included inside the parentheses, which are preceded by the **while** keyword rather than **for**.

while and do…while

- The same **three items** are still present, and they are still defined in the same order as they are in the **for** loop
- You still have to have an **initializer** defined before you can check whether it has reached the **exit-condition**;
- The **final-condition** is then run after the code inside the **while** loop.

while and do…while

- Demo 5: Let's have a look again at our cats list example, but rewritten to use a while loop:

```
// while
var i = 0;
while (i < cats.length) {
  if (i === cats.length - 1) {
    // last element;
    info += ' and ' + cats[i] + '.';
  } else {
    info += cats[i] + ', ';
  }

  i++;
}
```

while and do…while

- The do...while loop is very similar, but provides a variation on the while structure:

```
initializer
do {
  // code to run

  final-expression
} while (exit-condition)
```

while and do…while

- The **initializer** again comes first, before the loop starts.
- The **exit-condition** comes after everything else, wrapped in parentheses and preceded by a **while** keyword.
- In a do...while loop, the code inside the curly braces is always run **once** before the check (!!!)

while and do…while

- Demo 6: Let's rewrite our cat listing example again to use a do...while loop:

```
// do while loop
var i = 0;
do {
  if (i === cats.length - 1) {
    // last element;
    info += ' and ' + cats[i] + '.';
  } else {
    info += cats[i] + ', ';
  }

  i++;
} while (i < cats.length);
```

while and do…while - Overview

- **while** and **do…while** have very similar syntax
- The difference: the **exit-condition** comes after everything else, wrapped in parentheses and preceded by a while keyword
- With while and do...while — as with all loops — you must make sure that the initializer is incremented or, depending on the case, decremented, so that it eventually reaches the exit condition. If not, the loop will go on forever, and either the browser will force it to stop, or it will crash.
- This is called an **infinite loop**

Section 6

Which loop should i use?

Which loop should i use?

- For basic uses, for, while, and do...while loops are largely interchangeable.
- They can all be used to solve the **same problems**, and which one you use will largely depend on your **personal** preference — which one you find easiest to remember or most intuitive.

Which loop should i use?

- First **for** loop:

```
for (initializer; exit-condition; final-expression) {
  // code to run
}
```

Which loop should i use?

- while:

```
initializer
while (exit-condition) {
  // code to run

  final-expression
}
```

Which loop should i use?

- do…while:

```
1  initializer
2  do {
3    // code to run
4
5    final-expression
6  } while (exit-condition)
```

Which loop should i use?

- We would recommend **for**, at least to begin with, as it is probably the **easiest** for remembering everything
- The **initializer**, **exit-condition**, and **final-expression** all have to go neatly into the parentheses, so it is easy to see where they are and check that you aren't missing them.

Which loop should i use? - Overview

- For basic uses, **for**, **while**, and **do...while** loops are largely interchangeable.
- They can all be used to solve the same problems, and which one you use will largely depend on your personal preference — which one you find easiest to remember or most intuitive.
- **for** is the recommended loop for beginner (and when looping arrays)

Thank you

Q&A