JavaScript 1 - Module 1

Getting Started

Variables

Making Decisions

Loops

Solutions for JS1 Lesson 1.3 Making Decisions Exercises

Exercise 1

Make a variable myNumber with a number value of 10.

a) Make an if statement to see if the value of myNumber is equal to 10.

If so, console log out "Bingo, the number is 10".

b) Add an else statement that console logs out "Aww, the number wasn't 10". What is console logged out now?

c) Change myNumber to something other than 10. What is console logged out now?

```
// Exercise 1
console.log ("Exercise 1a");

var myNumber = 10;
if (myNumber === 10) {
    console.log ("Bingo, the number is 10");
}

console.log ("Exercise 1b + 1c");

myNumber = 12; // For c
if (myNumber === 10) {
    console.log ("Bingo, the number is 10");
} else {
    console.log ("Aww, the number wasn't 10");
}
```

Exercise 2

Make a (boolean) variable named apple valued true and another variable named orange valued false.

a) Make an if statement that compares the two and console logs out "You cannot compare apples and oranges" as long as the two variables are not equal.

b) Add an else statement that console logs out "Hm, it seems apples and oranges are the same, after all".

c) Change the orange variable's value to true. What is console logged out?

d) Then change the apple variable's value to false. Now, what is console logged out?

```
// Exercise 2
console.log ("Exercise 2a");
var apple = true;
var orange = false;

if (apple !== orange) {
    console.log("You cannot compare apples and oranges");
}

console.log ("Exercise 2b, c, d");
orange = true; // c
// apple = false; // d
if (apple !== orange) {
    console.log("You cannot compare apples and oranges");
} else {
    console.log("Hm, it seems apples and oranges are the same, after all");
}
```

Exercise 3

Ask the user to input their name.

If the name is set display a message saying "Hello, [name]".

If nothing was entered display a message saying "Awww!"

> Tip: Use prompt() to allow input

What happens if the user presses "Cancel"?

```
console.log ("Exercise 3");
let inputName = prompt("What is your name?");

if (inputName !== "") {
    alert ("Hello, " + inputName);
} else {
    alert ("Awww!");
}
// Cancel gives "Hello, null"
```

Exercise 4

Make a variable myAge and give it a numeric value (eg. your age).

Make an `if...else if...else` statement that console logs out the following scenarios:

```
If myAge is less than 0, it says "That's not possible".
If myAge is less than 18, it says "You are juvenile".
If myAge is less than 30, it says "You are still young".
If myAge is less than 50, it says "You aren't exactly young anymore, are you?"
If myAge is less than 70, it says "You're getting really old, aren't you?"
If myAge is less than 100, it says "How is retirement treating you?"
If myAge is 100 or more, it says "Centennial? Impressive...";
```

```
// Exercise 4

console.log ("Exercise 4");

var myAge = 50;

if (myAge < 0 || isNaN(Number(myAge))) { // Just an extra check
//if (myAge < 0) {
    console.log("That's not possible")
} else if (myAge < 18) {
    console.log("You are juvenile");
} else if (myAge < 30) {
    console.log("You are still young");
} else if (myAge < 50) {
    console.log("You aren't exactly young anymore, are you?");
} else if (myAge < 70) {
    console.log("You're getting really old, aren't you?");
} else if (myAge < 100) {
    console.log("How is retirement treating you?");
} else {
    console.log("Centennial? Impressive...");
}
```

Exercise 5

Ask the user input a number.

Check to see if it was really a number that was entered.

Display "Valid number entered" or "Not a valid number entered" back to the user.

Tip: Check the input using the `Number()` function and the `isNaN()` function.

```
// Exercise 5

console.log ("Exercise 5");

var number = prompt("Please, enter a number");
var test = Number(number);

if (isNaN(test)) {
    console.log (number + " is not a number: Not a valid number entered");
} else {
    console.log (number + " is indeed a number: Valid number entered");
}
```

Exercise 6

Ask the user input a number.

Display the number and whether it is odd or even, back to the user.

> Tip: To check if a number is odd or even you may use reminder division where you check the reminder when dividing by 2; if the reminder is 1, then the number is odd; if the reminder is 0, then the number is even.

```
console.log ("Exercise 6");
var number = prompt("Please, enter a number");

if (number % 2 == 0) {
    console.log (number + " is an even number");
} else {
    console.log (number + " is an odd number");
}
```

Exercise 7

Extend the code from Exercise 3, to check if nothing was entered or the user just pressed cancel.

> Tip: If the user clicks "OK" on a prompt(), the input value is returned as a string. If the user clicks "cancel", null is returned. If the user clicks OK without entering any text, an empty string is returned.

```
console.log ("Exercise 7");

let inputName = prompt("What is your name?");

if (inputName === "" || inputName === null) {
    alert ("Awww!");
} else {
    alert ("Hello, " + inputName);
}
// NOTE: according to documentation, prompt() should return null when cancel is pressed
// Beware that if inputName is decleard with var, and not let, then null is returned as a string
```

Exercise 8

Make a variable fruit and give it the value "pear", and an undefined variable named output.

Make a switch statement that uses fruit as it's expression and the following cases:

```
apple: give output the value "An apple a day..."
orange: give output the value "What to do with all the orange peel?"
pear: give output the value "Delicious, from a tree or a can."
banana: give output the value "What, are you a Minion?"
mango: give output the value "Mangos are weird..."
```

a) Console log out the output variable. What is console logged out?

b) Make a default case that gives output the value "That's not a fruit, at least not one I've heard of."

c) Change the original fruit variable to the value "kiwi". What is console logged out now?

```
// Exercise 8

console.log ("Exercise 8 combined");

var fruit = "kiwi";
var output;
switch (fruit) {
    case "apple":
        output = "An apple a day...";
        break;
    case "orange":
        output = "What to do with all the orange peel?";
        break;
    case "pear":
        output = "Delicious, from a tree or a can.";
        break;
    case "banana":
        output = "What, are you a Minion?"
        break;
    case "mango":
        output = "Mangos are weird...";
        break;
    default:
        output = "That's not a fruit, at least not one I've heard of.";
}
console.log(output);
```

Exercise 9

Ask the user to enter a month.

a) Make a switch statement that takes the input and console logs out one of the following.

```
If the input is December, January or February, then console log: "It's Winter!".
If it's March, April or May, then log out "It's Spring!".
If it's June, July or August, then log out "It's Summer!".
If it's September, October or November, then log out "It's Autumn!".
```

b) Make a default case that logs out "That's not a month...". Now, enter "may" - and not "May" as input; what is logged out?
c) Make sure that the input no longer is case sensetive; so as an example it doesn't matter if the input is "May" or "may", it displays "It's Spring!" either way.

> Tip look at str.toLowerCase() rather than adding cases.

```javascript
// Exercise 9

console.log ("Exercise 9 combined");
let month = prompt("Enter a month: ");
month = month.toLowerCase();
switch (month) {
    case "december":
    case "january":
    case "february":
        console.log("It's Winter!"); break;
    case "march":
    case "april":
    case "may":
        console.log("It's Spring!"); break;
    case "june":
    case "july":
    case "august":
        console.log("It's Summer!"); break;
    case "september":
    case "october":
    case "november":
        console.log("It's Autumn!"); break;
    default:
        console.log("That's not a month...");
}
```

Exercise 10

Ask the user to input a number representing the voltage needed for an appliance (typical 1.5, 6, 12, 24, 110, 230, 480, etc).

If the number is between 220 and 240, then display an alert saying "This can run on the usual grid".

If not display "This need an inverter or something".

> Tip to make an expression for between you can check if x is equal or more than the lowest and at the same time x is also less than or equal to the highest in the range.

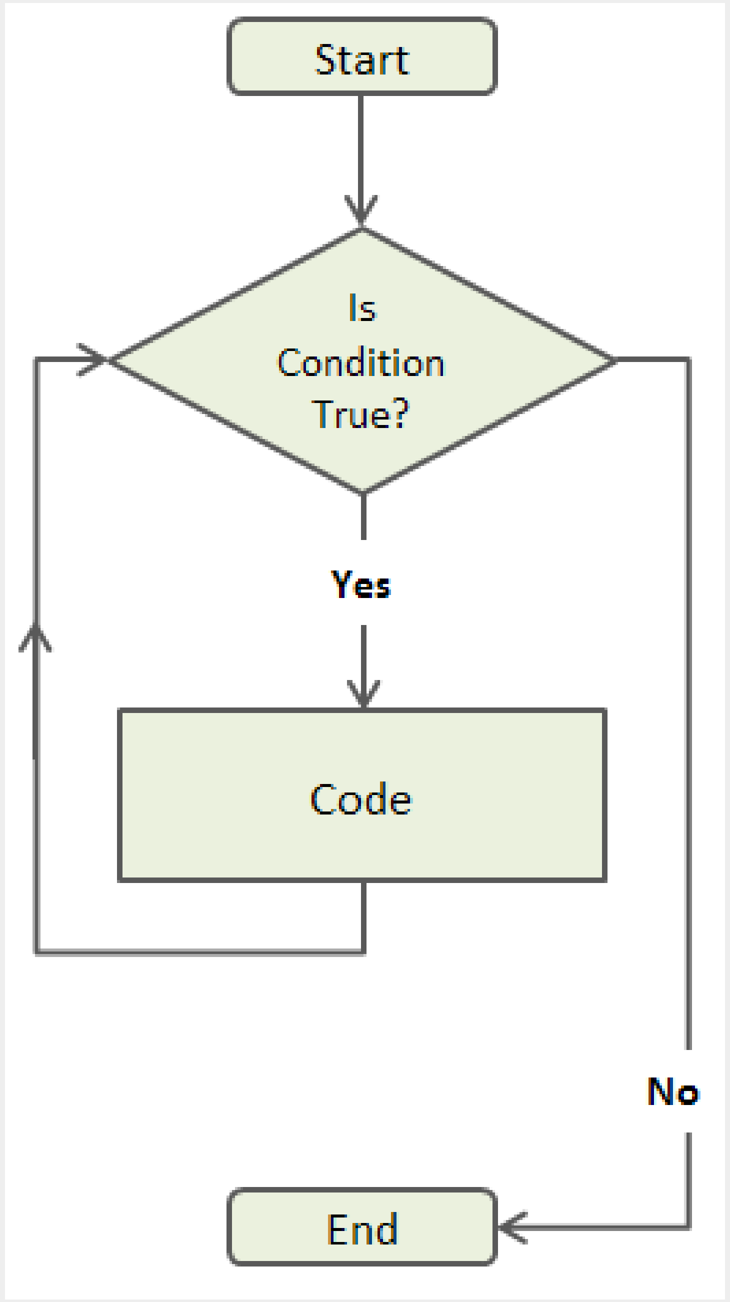
```javascript
// Exercise 10

console.log ("Exercise 10");

let chosenVoltage = prompt("What voltage does your appliance need?");
chosenVoltage = parseInt(chosenVoltage); // Convert mixed answer to number

if (isNaN(chosenVoltage)) {
    // chosenVoltage still isn't a number after conversion
    alert("You need to input a number");
} else {
    // chosenVoltage is a number after conversion
    if (chosenVoltage >= 220 && chosenVoltage <= 240) {
        // chosenVoltage is between 220 and 240
        alert ("This can run on the usual grid");
    } else {
        // chosenVoltage is NOT between 220 and 240
        alert ("This need an inverter or something");
    }
}
```

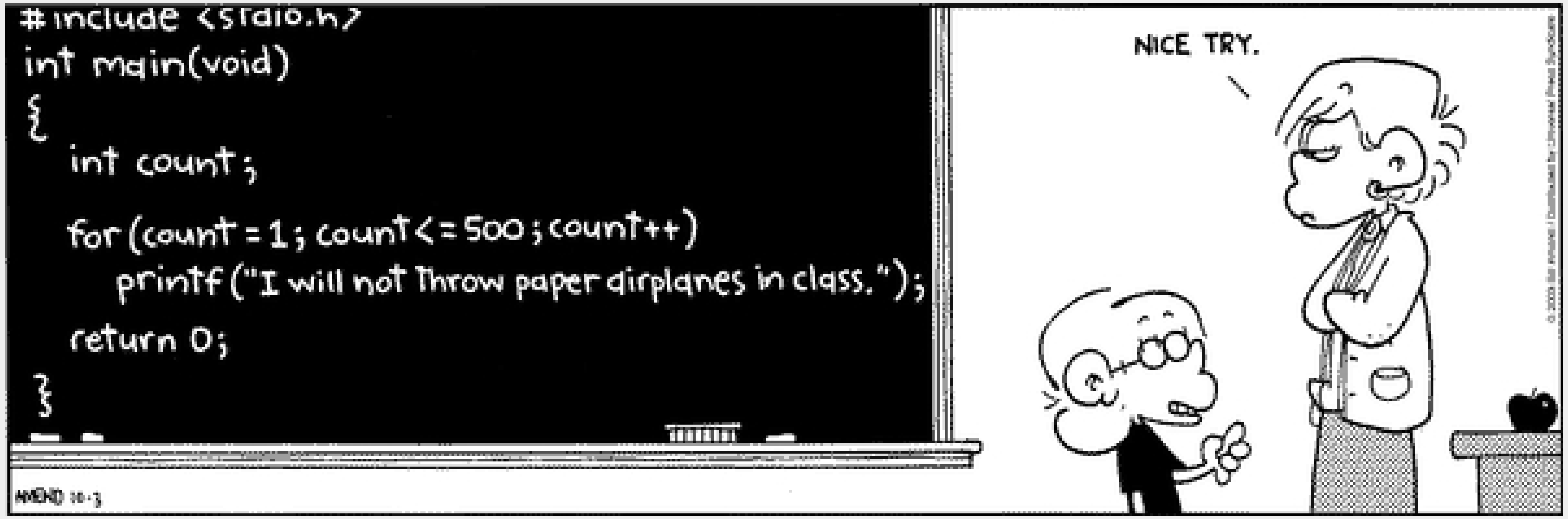
Loops



Loops

Programming languages are very useful for rapidly completing repetitive tasks, from multiple basic calculations to just about any other situation where you've got a lot of similar items of work to complete.

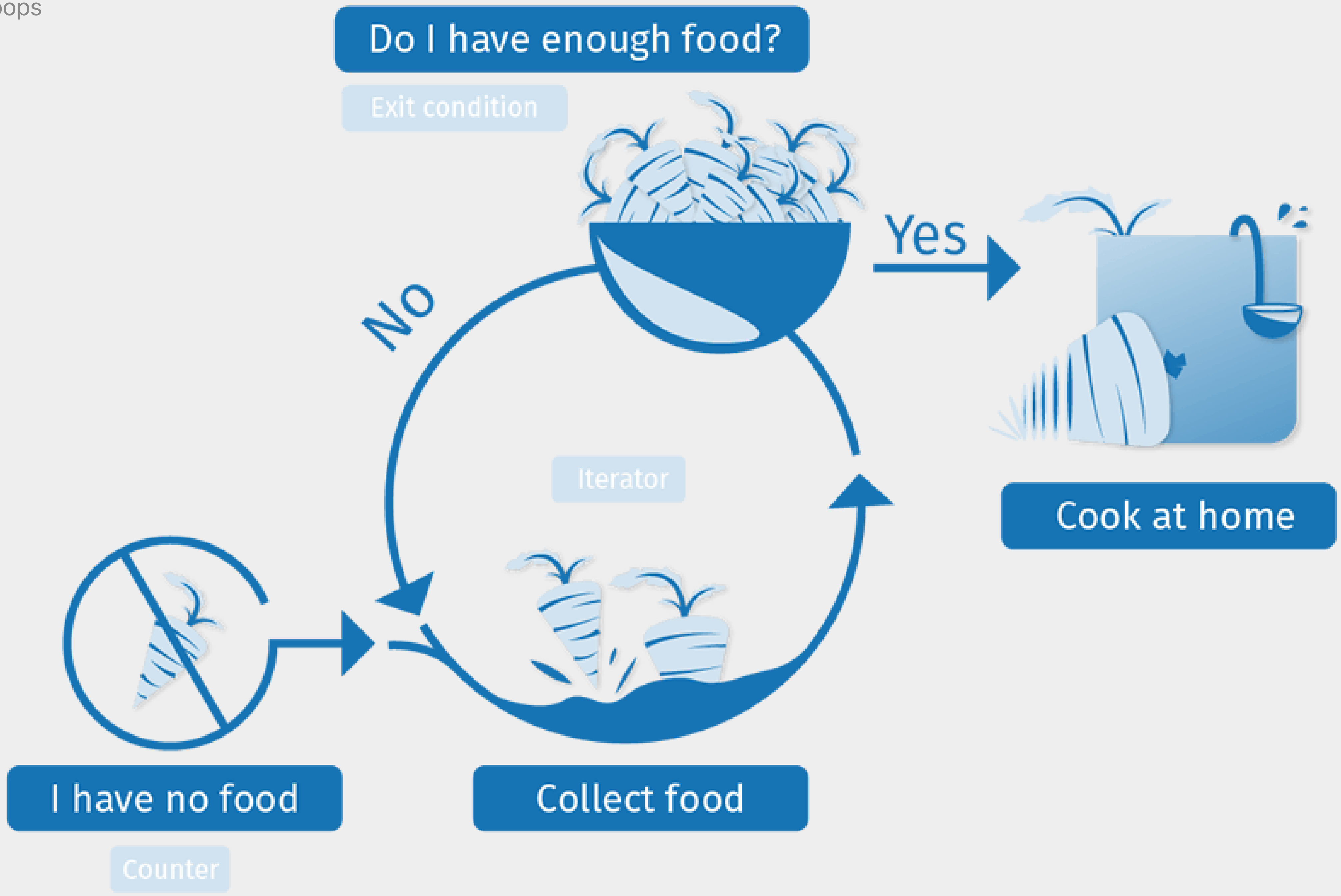
Programming **loops** are all to do with doing the same thing over and over again, which is termed *iteration* in programming speak.



```
#include <stdio.h>
int main(void)
{
  int count;
  for(count = 1; count <= 500; count++)
    printf("I will not Throw paper airplanes in class.");
  return 0;
}
```

A loop usually has one or more of the following features:

- A **counter**,
  which is initialized with a certain value — this is the starting point of the loop.
- A **condition**,
  which is a true/false test to determine whether the loop continues to run, or stops — usually when the counter reaches a certain value.
- An **iterator**,
  which generally increments the counter by a small amount on each successive loop until the condition is no longer true.


Do I have enough food?
Exit condition
No
Yes
Iterator
Cook at home
I have no food
Counter
Collect food

Source

In pseudocode, the illustration on the last slide would look something like the following:

```
loop(food = 0; foodNeeded = 10) { // food is the Counter

    if (food >= foodNeeded) { // This is the Condition

        exit loop;
        // We have enough food; let's go home

    } else {

        food += 2; // 2 is the Iterator
        // Collect 2 more food

        // loop will then run again

    }
}
```

Different Kinds of Loops

JavaScript supports different kinds of loops:

- `for` loops through a block of code a number of times
- `while` loops through a block of code while a specified condition is true
- `do...while` also loops through a block of code while a specified condition is true
- `for...in` loops through the properties of an object
- `for...of` loops through the values of an iterable object

The While Loop

The `while` loop loops through a block of code as long as a specified condition is true.

Syntax:

```
while (condition) {
  // code block to be executed
}
```

> Note: JavaScript statements can be grouped together in code blocks, inside curly brackets `{...}`.
>
> The purpose of code blocks is to define statements to be executed together.

The While Loop, an example

In the following example, the code in the loop will run, over and over again, as long as a variable (`i`) is less than 10:

```
var i = 0; // Counter starts at 0

while (i < 10) {  // Condition: loop as long as i is less than 10

    console.log("The number is " + i);
    i++; // Iterate the counter by 1 for each round in the loop

}
```

> If you forget to increase the variable used in the condition, the loop will never end, you've entered an infinite loop. This *will* crash your browser.

Codepen

The Do...While Loop

The `do...while` loop is a variant of the `while` loop.

This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

Syntax:

```
do {
  // code block to be executed
}
while (condition);
```

The Do...While Loop, an example

The example below uses a `do...while` loop. The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested:

```
var i = 0;

do {
  console.log( "The number is " + i );
  i++;
}
while (i < 10);
```

> If you change the counter to `var i = 100;` the do...while loop will run exactly once.

Codepen

Reminder: Do not forget to increase the variable used in the condition, otherwise the loop will never end!

The For Loop

The for loop has the following syntax:

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

- An **initializer**
  this is usually a variable set to a number, which is incremented to count the number of times the loop has run. It is also sometimes referred to as a counter variable.

> Note: The initializer may be set before the loop, but normally you set it at the start of the `for`.

For loop syntax, cont.

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

- A **condition**
  this defines when the loop should stop looping. This is generally an expression featuring a comparison operator, a test to see if the exit condition has been met.
- A **final-expression**
  this is always evaluated (or run) each time the loop has gone through a full iteration. It usually serves to increment (or in some cases decrement) the counter variable, to bring it closer to the point where the condition is no longer true.

The For loop, an example

```
for (var i = 0; i < 5; i++) {
    console.log("The number is " + i);
}
```

The initializer sets a variable before the loop starts (`var i = 0`).

The condition for the loop to run is set (`i` must be less than 5).

The final-expression increases the value of `i` by 1 (i++) each time the code block in the loop has been executed.

The For loop, another example

Loops can also be made to run "backwards":

```
for (var i = 10; i >= 0; i--) {
    console.log("The number is " + i);
}
```

The initializer sets a variable before the loop starts (`var i = 10`).

The condition for the loop to run is set (`i` must be greater than or equal to 0).

The final-expression **decreases** the value of `i` by 1 (i--) each time the code block in the loop has been executed.

Codepen

The For loop, yet another example

Loops are perfect for running through (looping over) an array. (Arrays are used to store multiple values in a single variable.)

```
const cats = ['Sansa', 'Arya (rip)', 'Bran'];
let info = 'My cats are called ';

for (let i = 0; i < cats.length; i++) {
  info += cats[i] + ', ';
}

console.log(info); // My cats are called Sansa, Arya (rip), Bran,
```

> More on arrays in lesson 2.2

Fixing the comma issue in that last example?

```
const cats = ['Sansa', 'Arya (rip)', 'Bran'];
let info = 'My cats are called ';

for (let i = 0; i < cats.length; i++) {
    if (i === cats.length - 1) {
        // Special case for last in list
        info += "and " + cats[i] + ".";
    } else {
        // For all others, just list them as usual
        info += cats[i] + ', ';
    }
}

console.log(info); // My cats are called Sansa, Arya (rip), and Bran.
```

> Note: For convenience I use the Oxford comma in this example.

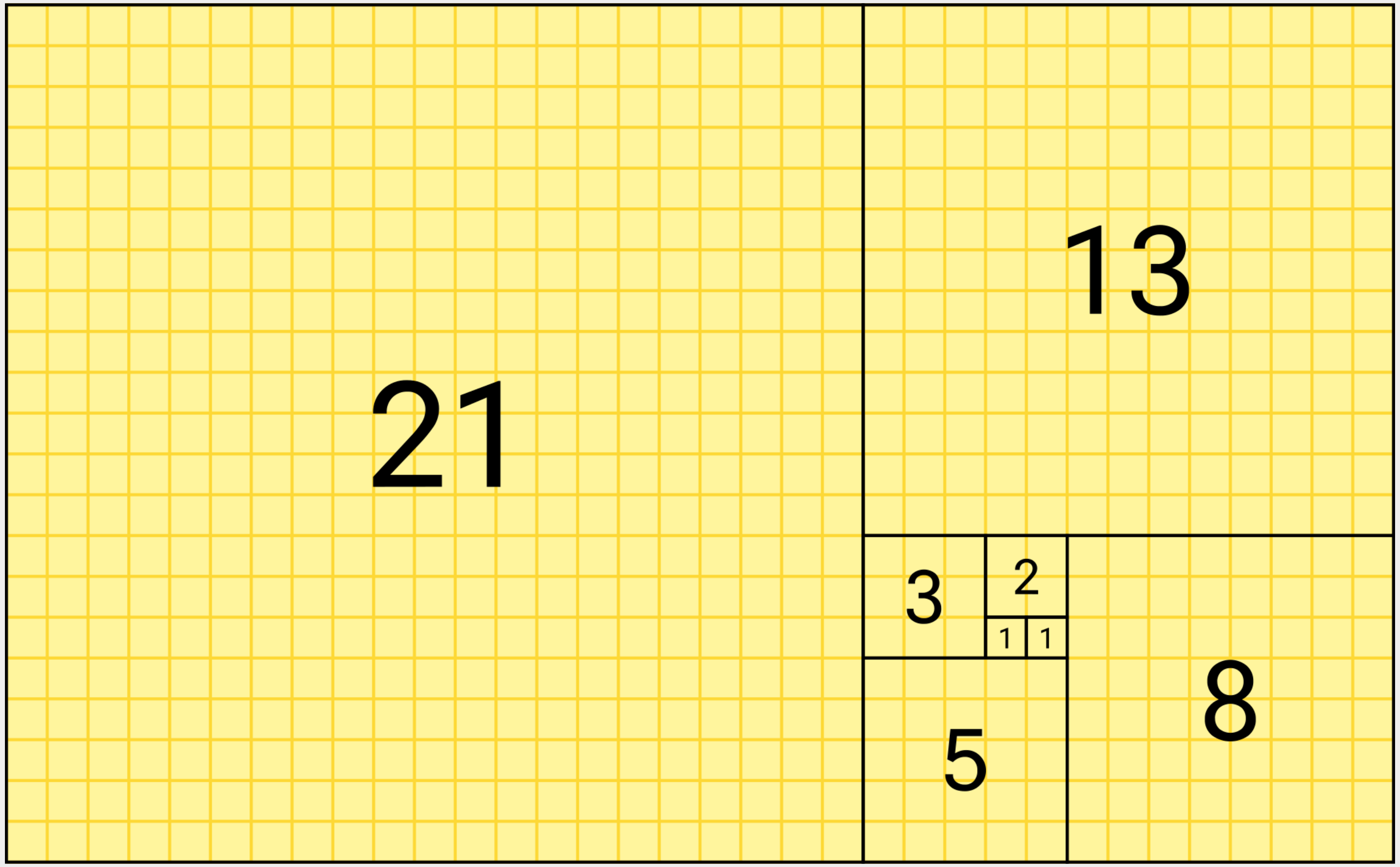
Fibonacci numbers

In mathematics, the Fibonacci sequence is a sequence in which each number is the sum of the two preceding ones. Numbers that are part of the Fibonacci sequence are known as Fibonacci numbers, commonly denoted F_n. The sequence commonly starts from 0 and 1, although some authors start the sequence from 1 and 1 or sometimes (as did Fibonacci) from 1 and 2. Starting from 0 and 1, the sequence begins

```
0, 1, 1, 2, 3, 5, 8, 13, 21, 34
```

Fibonacci numbers are strongly related to the golden ratio, and appear unexpectedly often in mathematics as well as nature.

Source



A tiling with squares whose side lengths are successive Fibonacci numbers: 1, 1, 2, 3, 5, 8, 13 and 21

Using a loop to find Fibonacci numbers

```
let fib = [0, 1];

for (let i = 2; i < 10; i++) {
  fib[i] = fib[i - 2] + fib[i - 1];
}

console.log(fib);

// > Array(10) [ 0, 1, 1, 2, 3, 5, 8, 13, 21, 34 ]
```

Comparing the `while` and the `for` loop

In a typical `while` loop, initialization is before the loop. And you need to iterate the counter variable inside the loop:

```
initialization;
while(condition)
{
    body
    decrement/increment variable
}
```

In a typical `for` loop, initialisation, condition and the iteration is done when starting the loop.

```
for(initialization; condition; decrement/increment)
{
    body
}
```

These two loops will yield excatly the same result:

```
let i = 10;
while( i > 5 ) {
    console.log("The i is " + i);
    i--;
}

for(let i = 10; i > 5; i--)  {
    console.log("The i is " + i);
}
```

The `for` loop, as you can see, is often a bit more compact than the `while` loop.

> Tip: Use the for loop when you need to run through a fixed set of items, and the while (or do...while) when you aren't sure how many times the loop should run.

The `break` statement

You have already seen the `break` statement used in an earlier chapter of this lesson. It was used to "jump out" of a `switch()` statement.

The `break` statement can also be used to jump out of a loop, to continue executing the code after the loop (if any):

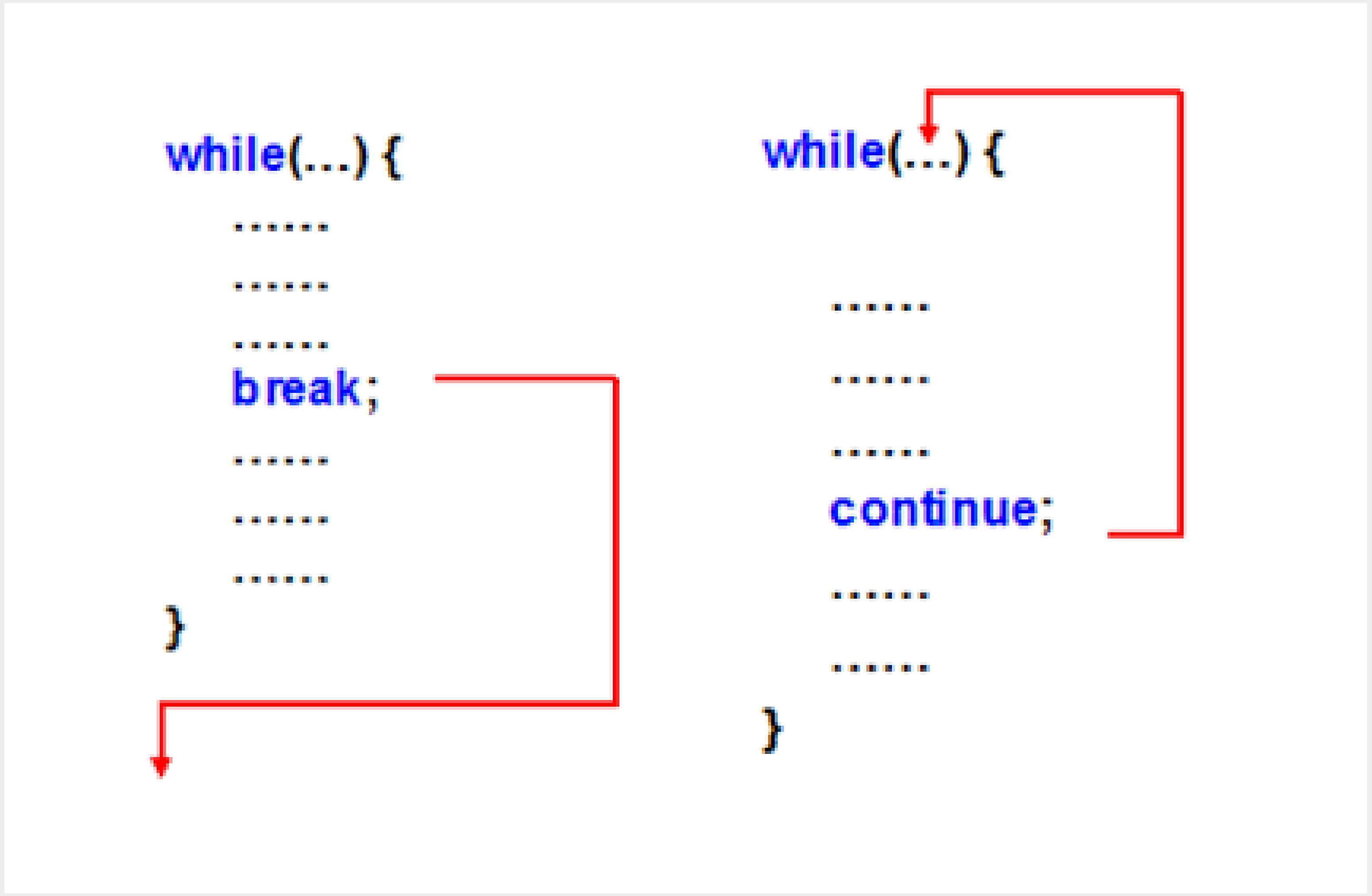
```
for (let i = 0; i < 10; i++) {
  if (i === 2) { break; }
  console.log ("The number is " + i);
}
console.log ("Done");
// Expected output:
// "The number is 0"
// "The number is 1"
// "Done"
```

The `continue` statement

The `continue` statement breaks *one iteration* (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

This example skips the value of 3:

```
let text = "";
for (let i = 0; i < 10; i++) {
  if (i === 3) { continue; }
  text += i + " ";
}
console.log (text);
// Expected output: "0 1 2 4 5 6 7 8 9 "
```



The break statement "jumps out" of the loop.

The continue statement "jumps over" one iteration in the loop.

The For...of Loop

The JavaScript `for...of` statement loops through the values of an iterable objects.

```
for (variable of iterable) {
    // code block to be executed
}
```

variable: For every iteration the value of the next property is assigned to the variable.

Variable can be declared with const, let, or var.

iterable: An object that has iterable properties.

`for...of` lets you loop over data structures that are iterable such as Arrays, Strings, Maps, NodeLists, and more.

For...of examples

```
var text = "";
var cars = ["BMW", "Volvo", "Mini"];
for (let car of cars) {
    text += car + " ");
}
console.log(text);
// expected output: "BMW Volvo Mini "
```

```
const array = ['a', 'b', 'c'];
for (const element of array) {
    console.log(element);
}
// expected output:
// "a"
// "b"
// "c"
```

For...of examples, `for...of` vs "normal" `for`

```
var text = "JavaScript";
var newText = "";
for (let letter of text) {
    newText += letter + " ";
}
console.log(newText);
// expected output: "J a v a S c r i p t "
```

```
var text = "JavaScript";
var newText = "";
for (let i = 0; i < text.length; i++) {
    newText += text[i] + " ";
}
console.log(newText);
// expected output: "J a v a S c r i p t "
```

These two loops will do exactly the same.

The For...in Loop

The JavaScript `for...in` loops through the properties of an object:

```
for (variable in object) {
    // code block to be executed
}
```

variable: A different property name is assigned to variable on each iteration.

object: Object whose non-Symbol enumerable properties are iterated over.

> Note: for...in should not be used to iterate over an Array where the index order is important.

For...in examples

```javascript
var person = {fname:"John", lname:"Doe", age:25};
var text = "";

for (var p in person) {
  text += person[p];
}
console.log (text); // expected output: "John Doe 25"
```

```javascript
const object = { a: 1, b: 2, c: 3 };

for (const property in object) {
  console.log(`${property}: ${object[property]}`);
}
// expected output:
// "a: 1"
// "b: 2"
// "c: 3"
```

> Note the use of backticks for making a string (or a template literal, to be precise)

Difference between for...of and for...in

Both `for...in` and `for...of` statements iterate over something. The main difference between them is in what they iterate over.

The `for...in` statement iterates over the enumerable properties of an object, in an arbitrary order. *Use for object and debugging*.

The `for...of` statement iterates over values that the iterable object defines to be iterated over. *Use for all iterable objects, like Arrays*.

Difference between for...of and for...in, example

```
var fruits = ["apple", "banana", "mango", "pear"];
for (let fruit of fruits) {
    console.log ("The fruit is " + fruit);
}
// expected output:
// "The fruit is apple"
// "The fruit is banana"
// "The fruit is mango"
// "The fruit is pear"
```

```
var fruits = ["apple", "banana", "mango", "pear"];
for (let fruit in fruits) {
    console.log ("The fruit is " + fruit);
}
// expected output:
// "The fruit is 0"
// "The fruit is 1"
// "The fruit is 2"
// "The fruit is 3"
```

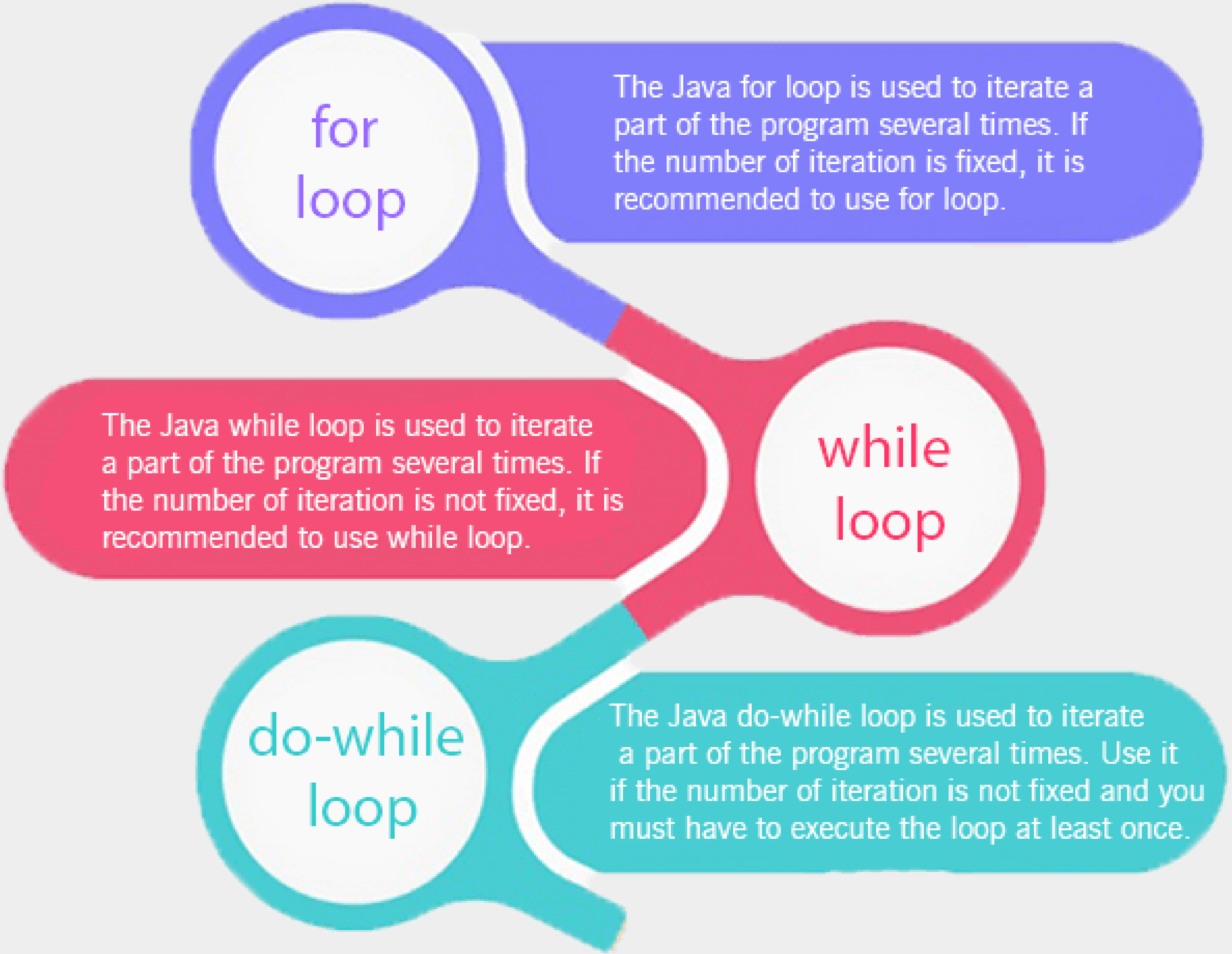
Singalong

```
let numberOfBeers = 99;

while (numberOfBeers > 0) {
    console.log(`${numberOfBeers} \
bottle${(numberOfBeers === 1)?"":"s"} of beer on the \
wall, ${numberOfBeers} \
bottle${(numberOfBeers === 1)?"":"s"} of beer. `);
    numberOfBeers--; // Take one down...
    console.log(`Take one down, pass it around, \
${(numberOfBeers !== 0)?numberOfBeers:"no more"} \
bottle${(numberOfBeers === 1)?"":"s"} \
of beer on the wall...`)
    console.log(``);
}

console.log(`No more bottles of beer on the wall, \
no more bottles of beer.`);
console.log(`We've taken them down and passed them around; \
now we're drunk and passed out!`);
```

Codepen



The Java for loop is used to iterate a part of the program several times. If the number of iteration is fixed, it is recommended to use for loop.

The Java while loop is used to iterate a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop.

The Java do-while loop is used to iterate a part of the program several times. Use it if the number of iteration is not fixed and you must have to execute the loop at least once.

Different loops, and their uses. Example from a Java site about loops, but it's just as relevant for JavaScript.

Sources and resourses

MDN:

Looping code

Loops and iteration

W3School:

JavaScript While Loop

JavaScript For Loop

JavaScript Break and Continue

Todos

GitHub Classroom

JS1 Lesson 1.4 Loops

Mollify

Read Loops, and do the Lesson Task.