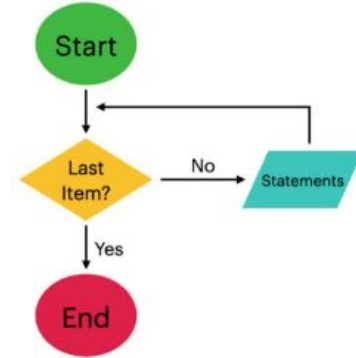# Web development basics

Loops, functions, data fetching

Loops (for)

In JavaScript, loops are used to execute a block of code repeatedly until a certain condition is met. There are several types of loops in JavaScript, including:

- **for** loops: The for loop is used to execute a block of code a specific number of times.

For Loop

For loop

```
for (let i = 0; i < 10; i++) {
    console.log(i);
}
```

for keyword → for

Condition:
- initialisation
- condition
- updater

Code to execute when the loop runs

Loops (while, do-while)

- **while** loops: The while loop is used to execute a block of code repeatedly while a certain condition is true.

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

- **do-while** loops: The do-while loop is similar to the while loop, but the code inside the loop is executed at least once, regardless of whether the condition is true or false.

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

Other Loops (foreach,map)

There are many other loops in js, each used for specific scenario, most common ones are:

- **foreach** - method allows you to execute a function on each element of an array. This method does not return a new array, it simply loops through the array and performs the function on each element.

```javascript
let numbers = [1, 2, 3, 4, 5];
numbers.forEach(function(num) {
  console.log(num);
});
```

- **map** - method is similar to forEach(), but it returns a new array with the results of the function applied to each element.

```javascript
let numbers = [1, 2, 3, 4, 5];
let doubledNumbers = numbers.map(function(num) {
  return num * 2;
});
console.log(doubledNumbers);
```

Classwork - Loops

- Using a for loop to print out the numbers 1 through 10
- Using a forEach loop to print out the elements of an array
- Using a map function to create a new array with the square of each element in an original array

Classwork - Loops (solution)

```javascript
for (let i = 1; i <= 10; i++) {
    console.log(i);
}
```

```javascript
let numbers = [1, 2, 3, 4, 5];
let squares = numbers.map(function(number) {
    return number * number;
});
console.log(squares);
```

```javascript
let fruits = ["apple", "banana", "orange"];
fruits.forEach(function(fruit) {
    console.log(fruit);
});
```
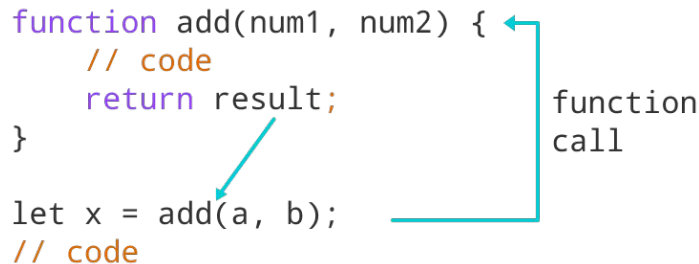
Classwork - Loops (solution 2)

```javascript
let fruits = ["apple", "banana", "orange"];
fruits.forEach(fruit => console.log(fruit));

let numbers = [1, 2, 3, 4, 5];
let squares = numbers.map(number => number * number);
console.log(squares);
```

Functions

```
function add(num1, num2) {
    // code
    return result;
}

let x = add(a, b);
// code
```

function call

In JavaScript, a function is a block of code that can be reused multiple times. Functions can take **input** (called parameters) and return **output** (the result of the function).

To define a function, you use the keyword "**function**" followed by the **function name**, a set of **parentheses** for any parameters, and a set of **curly braces** for the code that will be executed when the function is called.

Functions 2

```
function add(a, b) {
  return a + b;
}
```

```
let sum = add(3, 4);
console.log(sum); // Output: 7
```

```
const sum = (a,b) => a+b;
console.log(sum(4,5)); // Output: 9
```

What is "this"?

https://dmitripavlutin.com/gentle-explanation-of-this-in-javascript

Introduction to the fetch() API

fetch() is a **JavaScript API** that provides an **interface** for **fetching** resources (including across the network). It's purpose is to retrieve and handle data from an API or any other web resource. The fetch() API uses **Promises** and can be used in modern **browsers** and in **Node.js** environments.

With fetch(), you can make **HTTP requests**, such as **GET** and **POST**, to retrieve or send data to a remote resource. The API returns a Promise that resolves to the **Response object** representing the response to your request. This response object can then be used to access the data returned by the API, typically in **JSON** format.

Making a Basic fetch() Request (Promise chain)

In this example, the **fetch**() function is called with the API endpoint **URL** as its argument. The function returns a Promise that resolves to the Response object. The .**then**() method is used to extract the **JSON** data from the response and log it to the console. The .**catch**() method is used to handle any errors that may occur during the fetch operation.

Note: It's important to check the ok property of the Response object to ensure that the API returned a successful response before attempting to extract the data.

```javascript
fetch('https://api.example.com/data')
  .then(response => {
    if (!response.ok) {
      throw new Error('Network response was not ok');
    }
    return response.json();
  })
  .then(data => {
    console.log(data);
  })
  .catch(error => {
    console.error('There was a problem with the fetch operation:', error);
  });
```

Handling Response Data

The Response object can contain different types of data, such as **JSON**, **text**, **blob**, and others. To handle different types of data, you can use methods like .**json**(), .**text**(), and .**blob**() to extract the data in the format you need.

In this example, the .**text**() method is called to extract the response **data as text**. The text data can then be used in your application.

It's important to understand the format of the data returned by the API and use the appropriate method to extract and handle the data.

```javascript
fetch('https://api.example.com/data')
  .then(response => {
    if (!response.ok) {
      throw new Error('Network response was not ok');
    }
    return response.text();
  })
  .then(text => {
    console.log(text);
  })
  .catch(error => {
    console.error('There was a problem with the fetch operation:', error);
  });
```

Using fetch() with Async/Await

In this example, the getData() function is declared as **async**, which allows the use of the **await** keyword inside the function. The fetch() function is called and the result is assigned to the response variable. **The await keyword is used to wait for the response to be received before proceeding with the rest of the code.**

Using **async/await** to handle the fetch() response is a cleaner and easier to understand approach compared to using **Promises**. It makes the code more readable and reduces the amount of code needed to handle the response. The code is also easier to understand, especially for developers who are new to JavaScript or Promises.

```javascript
async function getData() {
  try {
    const response = await fetch('https://api.example.com/data');
    if (!response.ok) {
      throw new Error('Network response was not ok');
    }
    const data = await response.json();
    console.log(data);
  } catch (error) {
    console.error('There was a problem with the fetch operation:', error);
  }
}

getData();
```

Classwork for async/await

Task: Display the titles of the latest 10 articles from a mock news API using async/await and fetch()

Instructions:

1) Use the following API endpoint to retrieve the latest 10 articles:
   https://jsonplaceholder.typicode.com/posts
2) Create a function called getArticles that will make a GET request to the API endpoint.
3) Inside the getArticles function, use fetch() to make a GET request to the API endpoint and store the response in a variable called response.
4) Use the await keyword to wait for the response to be received before proceeding with the rest of the code.
5) Check if the response was successful (status code 200) by calling response.ok. If it's not successful, throw an error with a message "Network response was not ok".
6) Extract the JSON data from the response by calling response.json() and store the result in a variable called data.
7) Use a for loop to iterate over the data array and log the title property of each article to the console.
8) Call the getArticles function to start the process.

Classwork for async/await - solution

```javascript
async function getArticles() {
  try {
    const response = await fetch('https://jsonplaceholder.typicode.com/posts');
    if (!response.ok) {
      throw new Error('Network response was not ok');
    }
    const data = await response.json();
    for (let i = 0; i < 10; i++) {
      console.log(data[i].title);
    }
  } catch (error) {
    console.error('There was a problem with the fetch operation:', error);
  }
}

getArticles();
```

https://www.prakse.lv/event/list/1/1/0/0/0/0/0

Homework

Task 1: Write a function called multiply that takes in two parameters, a and b, and returns the product of the two numbers. Call the function with different numbers as arguments and use console.log() to print out the result.

Task 2: Write a function called isEven that takes in a number as a parameter and returns a boolean value indicating whether or not the number is even. Use a for loop to iterate from 1 to 10 (inclusive) and call the isEven function for each number. Use console.log() to print out the number and whether or not it is even (i.e. "2 is even" or "3 is odd").

Optional Find any free API, create get/post/put/patch/delete requests with it.