JavaScript Functions

Funktionen werden benötigt um versch. Anweisungen zu bündeln und diese eben an beliebiger Stelle aufzurufen.

Functions are used to **avoid repeating code**. Functions are used to **make code reusable**. Functions are used to make code **easier to read and understand**

Eine Funktion ist ein Block von Code, der eine bestimmte Aufgabe ausführt. Eine Funktion wird durch einen Namen definiert, gefolgt von Klammern (), und einem Block von Code innerhalb von geschweiften Klammern {}. Der Name der Funktion wird verwendet, um die Funktion aufzurufen.

Beispiel für eine FUNKTION

a ( ); damit rufen wir die Funktion auf. Beachte!! => Funktionen werden nur ausgeführt, wenn die aufgerufen werden.
Man kann eine Funktion an beliebiger Stelle aufrufen bevor ich wir die erstellt haben z.B oben drüber.

function a (was in Klammern steht heißt Paramter) {
let variableEins = 1
console.log (variableEins);} **=> hier wird die Funktion erstellt**

a ( ); damit rufen wir die Funktion auf.

To call a function, use the name of the function followed by parentheses ().

Function Body

The code inside the curly braces {} is called the function body. The function body is executed when the function is called.

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

sayHello(); // Hel
```

Function Scope

Variables that are defined inside a function are only accessible inside the function. These variables are said to have function scope.

```
function sayHello() {
  let message = "Hello";
  console.log(message);
}

sayHello(); // Hello
console.log(message); // ReferenceError: message is not defined
```

Function Parameters

A function can have parameters. Parameters are variables that are used to pass data into a function. Parameters are defined inside the parentheses () when the function is defined.

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

Function Arguments

Arguments are the actual values that are passed into a function when it is called. Arguments are passed into a function inside the parentheses `()` when the function is called.

```
greetPerson("John");
```

Output: Hello John

Arguments can be variables, values, or expressions.

```
let name = "Alice";
greetPerson(name);
greetPerson("John");
greetPerson("John" + " Doe");
```

Arguments order

```
```js
function greetPerson(name, greeting) {
  console.log(greeting + " " + name);
}

greetPerson("John", "Hello"); // Hello John
greetPerson("Hello", "John"); // John Hello
```
```

Default Parameters

A function can have default parameters. Default parameters are used when no argument is passed into the function or when an argument is passed into the function with a value of `undefined`.

```
function greetPerson(name = "John") {
  console.log("Hello " + name);
}

greetPerson(); // Hello John
greetPerson("Alice"); // Hello Alice
greetPerson(undefined); // Hello John
```

Return a value from a function

A function can return a value. The value that is returned can be used in other parts of the program. The `return` keyword is used to return a value from a function.

```
function addNumbers(num1, num2) {
  return num1 + num2;
}

addNumbers(2, 3); // 5
addNumbers(5, 7); // 12
addNumbers(); // NaN (Not a Number) because no arguments were passed

function addNumbers(num1, num2) {
  num1 + num2;
}

addNumbers(2, 3); // undefined because no value was returned
```

Function data type

A function is a data type. A function can be assigned to a variable, passed into a function as an argument, and returned from a function.

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

let sayHello2 = sayHello;

sayHello2(); // Hello
```

In JavaScript, functions are a type of object. This means that in addition to being callable (i.e., you can invoke them as functions), they can also have properties and methods like any other object. This is possible because functions are instances of the `Function` constructor, and as objects, they inherit properties and methods from the `Function.prototype` object.

Here's an example to illustrate this concept:

```
function greet(name) {
  console.log(`Hello, ${name}!`);
}

// You can call the function as usual
greet("Alice"); // Output: "Hello, Alice!"

// Functions are also objects, so you can add properties to them
greet.counter = 0;

// Access and use the property
console.log(greet.counter); // Output: 0

// Functions can even have methods
greet.sayGoodbye = function (name) {
  console.log(`Goodbye, ${name}!`);
};

// Call the method
```

```
greet.sayGoodbye("Bob"); // Output: "Goodbye, Bob!"
```

In the example above, `greet` is a function, but it is also an object. You can assign properties to it, such as `counter`, and you can even add methods to it, like `sayGoodbye`. This is because functions in JavaScript are "first-class citizens," which means they can be treated as values, assigned to variables, passed as arguments, and used as properties of objects.

This flexibility in JavaScript allows you to use functions in various ways, not just for their primary purpose of being callable.

Function Expressions

A function expression is a function that is assigned to a variable. Function expressions are not hoisted. Function expressions are not accessible before they are defined.

```
const sayHello = function () {
  console.log("Hello");
};

sayHello(); // Hello

// or we can do this:

function sayHello2() {
    console.log("Hello-2");
};

const sayHello3 = sayHello2;
```

Named and Anonymous Functions

Functions can be named or anonymous. Named functions are defined by a name followed by parentheses `()` and a block of code inside curly braces `{}`. Anonymous functions are defined by a variable followed by parentheses `()` and a block of code inside curly braces `{}`.

```
// Named function
function sayHello() {
  console.log("Hello");
}

// Anonymous function
const sayHello = function () {
  console.log("Hello");
};
```

Choose good names for functions

A function name should describe what the function does. A function name should be a verb or a verb phrase. A function name should be camelCase. A function name should not start with a number. A function name should not contain spaces.

```
// Good function names
function getFullName() {}
function getAge() {}
function getPerson() {}

// Bad function names
function fullName() {}
function age() {}
function person() {}
```

Using good names can also help if you are using an AI code assistant like copilot.