JavaScript Grundlagen

Übersicht

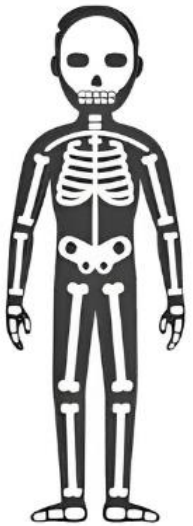
- Einleitung
- JS Berechnungen
- JS Datentypen
  - Number
  - String
  - Boolean
  - Spezielle Datentypen
- String Methoden
  - Length
  - Slice
  - toUpperCase, toLowerCase
  - indexOf, lastIndexOf
- Promp & Alert

Einleitung

JavaScript (JS) Wozu und Warum?

HTML

HTML the Skeleton

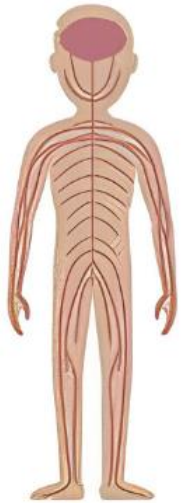


CSS

CSS the Skin

JS

Javascript the Brain



JavaScript (JS) Wozu und Warum?

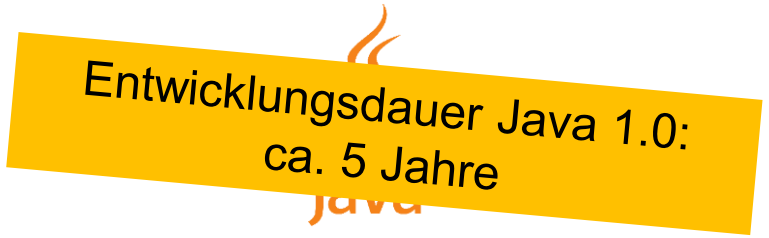
- **Webentwicklung:** JS wird hauptsächlich für die Entwicklung interaktiver Webseiten und Webanwendungen verwendet. Es ermöglicht die Erstellung von dynamischen Inhalten, interaktiven Benutzeroberflächen und die Manipulation des DOMs.
- **Client-seitige Funktionalität:** JS ermöglicht die Ausführung von Skripten direkt im Webbrowser des Clients. Dadurch können Benutzeraktionen verarbeitet, Formulardaten überprüft, Daten manipuliert und Animationen auf der Client-Seite erstellt werden.
- **Serverseitige Entwicklung:** Durch Plattformen wie Node.js kann JavaScript auch serverseitig eingesetzt werden. Dadurch wird die Kommunikation mit Datenbanken und allgemeine Backendentwicklung über JS möglich
- **Mobile Entwicklung:** Mit Frameworks wie React Native und Ionic kann JavaScript auch für die plattformübergreifende mobile Entwicklung genutzt werden.

JavaScript Historie

- **Frühe Entstehung**: JavaScript wurde 1995 von Brendan Eich bei Netscape entwickelt, um Webseiten interaktiver zu gestalten.
- **Standardisierung**: Die European Computer Manufacturers Association (ECMA) standardisierte die Sprache, was zur Veröffentlichung des ECMAScript-Standards führte.
- **Browser-Kriege**: In den 1990er Jahren trug der Wettbewerb zwischen Netscape und Microsoft zur Weiterentwicklung von JavaScript bei, wodurch es leistungsfähiger und vielseitiger wurde.
- **Moderne Bedeutung**: Durch die allgemeine Weiterentwicklung des Webs hat JavaScript an Bedeutung gewonnen und ist heute eine der wichtigsten Programmiersprachen für die Webentwicklung.

JavaScript vs. Java

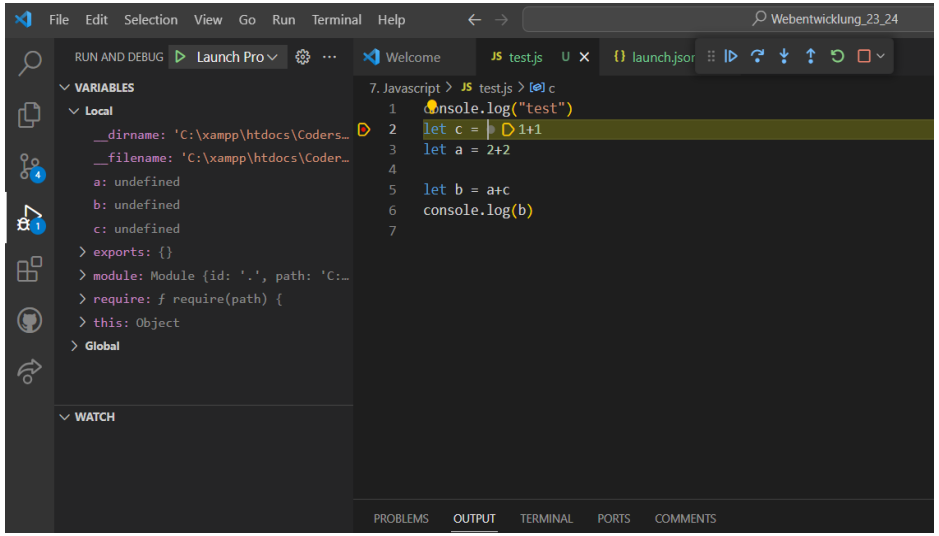
- **Typ der Programmiersprache:** JS ist eine Skriptsprache
- **Anwendungsbereich:** JS hauptsächlich für die Entwicklung von Webseiten und Webanwendungen verwendet wird
- **Ausführungsumgebung**: JS wird in einem Webbrowser oder auch auf der Serverseite mit Node.js ausgeführt
- **Typisierung:** JS ist dynamisch typisiert und die Typen werden zur Laufzeit bestimmt
- **Syntax und Struktur**: JS verwendet flexible und weniger strikte Syntax



- **Typ der Programmiersprache**: Java ist eine objektorientierte Programmiersprache
- **Anwendungsbereich:** Java wird oft für die Entwicklung von Desktop-Anwendungen, Serveranwendungen und Android-Apps verwendet
- **Ausführungsumgebung:** Java-Code wird von der Java Virtual Machine (JVM) ausgeführt
- **Typisierung:** Java ist statisch typisiert, Variablentypen müssen zur Kompilierungszeit festgelegt werden
- **Syntax und Struktur:** Java verwendet eine strikte Syntax mit strengen Regeln für Klassen und Vererbung

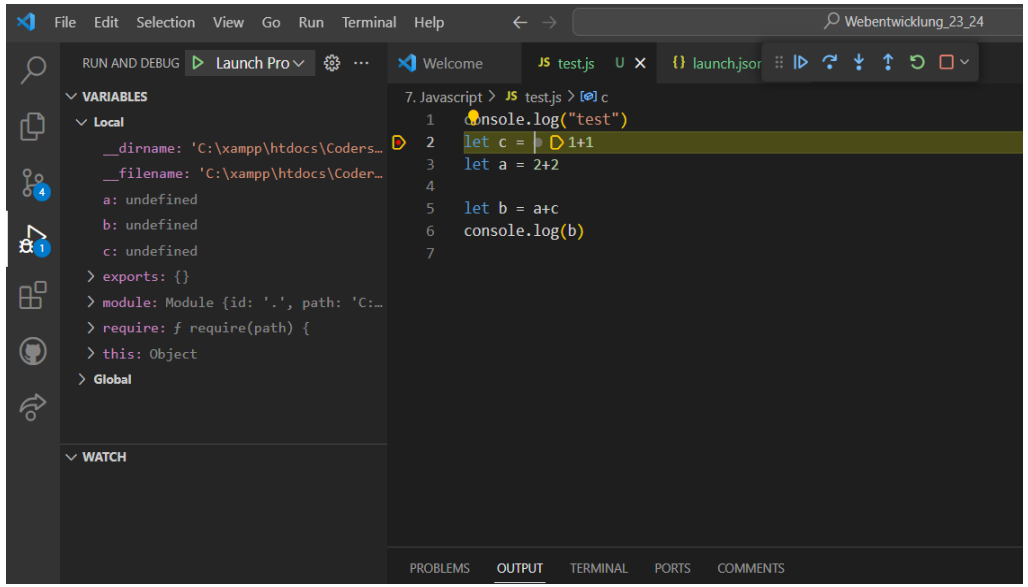
JavaScript in Visual Studio Code

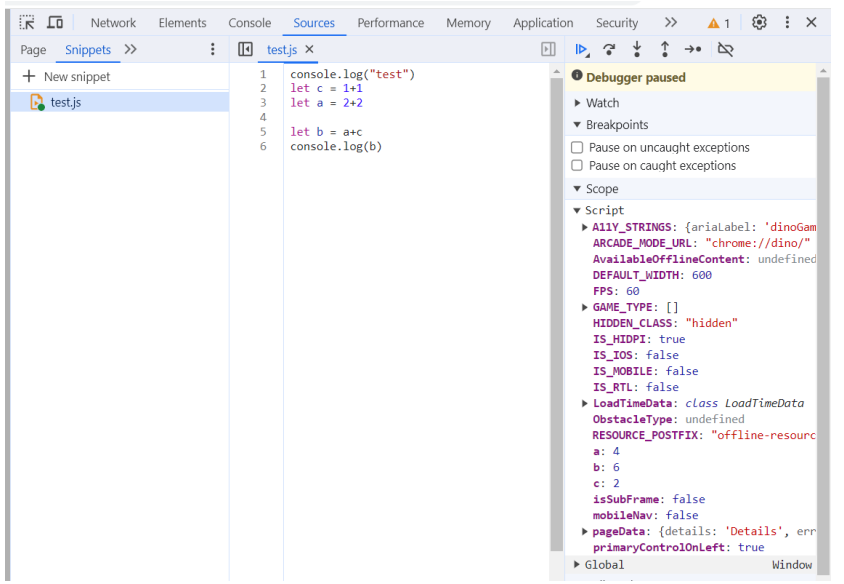
- Download der node.js Runtime
- Mit „F5“ wird das aktuelle file ausgeführt
- in der Debug Console wird das Ergebnis gelogged



https://nodejs.org/de

Ausführen des JS Codes





JS Berechnungen

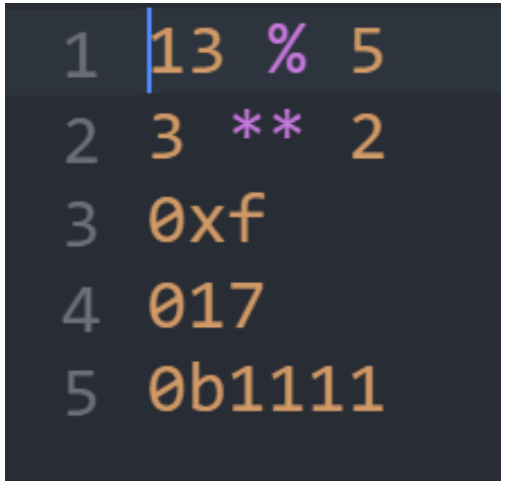
Grundrechenarten

- JavaScript kann rechnen, es kennt die 4 Grundrechenarten +, -, * , /
- Potenzieren wird mit ** erreicht Bsp: 10^4 = 10**4
- Es gilt die Regel Punkt-vor-Strich, Klammern ( ) können verwendet werden.
- JavaScript kann nicht nur mit ganzen Zahlen rechnen, sondern auch mit Kommazahlen.
- Zum Schreiben des "Komma" wird das Punkt-Zeichen (.) verwendet. Probiere die neben stehenden Berechnungen
- Vorsicht: Berechnungen sind nur bedingt präzise (bis ca. 15 Stellen)

```
1 1 + 1
2 3 - 5
3 10 / 2
4 2 + 2 * 3
5 (0.3-0.2-0.1)*10**4
```

Erweiterte Operationen und Zahlensysteme

- Es gibt noch zwei weitere Rechenoperatoren in JS
- Mit Modulo % erhält man den Rest einer Division
- Zum Potenzieren (hoch-Rechnen) kann das ** verwendet werden.
- Zahlen können durch voranstellen bestimmter Zeichen in anderen Zahlensystemen geschrieben werden
  - Hexadezimal (0x),
  - Oktalzahl (0)
  - Binärzahl (0b)



JS Variablen

Werte zwischenspeichern: Variablen

- Eine Variable ist ein Speicherort für beliebige Werte.
- Man kann Variablen Werte zuweisen, diese Auslesen und Überschreiben.
- Zum Zuweisen wird das = verwendet. Links davon steht der Name der Variable, rechts davon der Wert der gespeichert werden soll.
- Deklariert wird eine Variable mit den folgenden Schlüsselwörtern
  - var (vermeiden)
  - let
  - const
- Die Deklaration kann **vor** oder **während** der ersten Wertzuweisung geschehen
- Zur Verwendung wird die Variable an gewünschter Stelle einfach eingesetzt.
- Am Zeilenende setzt man in JavaScript für gewöhnlich ein Semikolon ;

```
1 let a = 2 * 3;
2 let b;
3 b = a + 1;
4 a = 0;
5 b;
6 a;
```

Deklaration mit var und der Scope

- Variablen können mit dem Schlüsselwort **var** deklariert werden
- Das ist allerdings veraltet und sollte nicht mehr verwendet werden
- In JavaScript gibt es sogenannte Scopes, diese werden mit { } geöffnet und wieder geschlossen.
- Ein Scope hat den Zweck Code zu bündeln, und zu begrenzen. So sollten Variablen, die in einem Scope definiert wurden außerhalb nicht verfügbar sein.
- Bei var ist das nicht der Fall! Außerhalb definierte Variablen sind in einem Scope übrigens verfügbar

```
1 {
2     var d = 3;
3 }
4 d;
```

```
1 {
2     let e = 3;
3 }
4 e;
```

Unterschiede let vs var

Ressourcen: 7. Javascript/Codebeispiele/letvsvar

Scope

```
// Example 1: Scoping within a block
{
    var varVariable = 'This is var';
    let letVariable = 'This is let';
}

console.log(varVariable); // accessible outside the block
// console.log(letVariable); // not accessible outside the block
```

Redeklaration

```
// Example 2: Redeclaration
var varVariable = 'Initial value';
let letVariable = 'Initial value';

// Redeclaring the variables
var varVariable = 'New value'; // No error
// let letVariable = 'New value'; // Error: SyntaxError
```

Hoisting

```
// Example 3: Hoisting
console.log(hoistedVar); // undefined
var hoistedVar = 'I am hoisted';

// console.log(hoistedLet); // ReferenceError: Cannot access
let hoistedLet = 'I am not hoisted';
```

Scope in a loop

```
// Example 4: Scope in a loop
for (var i = 0; i < 3; i++) {
    setTimeout(function() {
        console.log('The value of i with var is: ' + i); // Prints 3 three
    }, 1000);
}

for (let j = 0; j < 3; j++) {
    setTimeout(function() {
        console.log('The value of j with let is: ' + j); // Prints 0, 1, 2
    }, 1000);
}
```

Konstanten

- Konstanten werden ähnlich wie Variablen verwendet, sie werden allerdings mit dem Schlüsselwort const definiert
- Man **muss** ihnen bei der Deklaration **einen Wert zuweisen**.
- Anders als Variablen kann der Wert einer Konstante nicht überschrieben werden.

```
1 const F = 12 * 2;
2 F;
3 F = 3;
4
5 let g = 12 * 2;
6 g;
7 g = 3;
8 g;
```

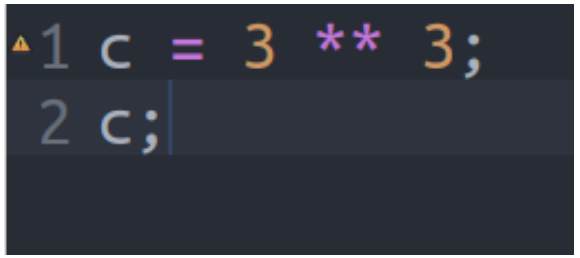
Namenskonvention

- Variablen Namen können aus folgenden Zeichen bestehen
  - Buchstaben
  - Unterstrichen
  - Dollarzeichen
  - Zahlen (wobei eine Zahl nicht am Anfang stehen darf)
- Umlaute und Sonderzeichen sollten vermieden werden
- Es ist üblich, Englische Bezeichnungen zu verwenden
- Variablen Namen die aus mehreren Wörtern bestehen in camelCase zu schreiben:
  - Alle Wortanfänge außer dem ersten mit Großbuchstaben, keine Leerzeichen
- Konstanten, deren Wert im Code festgelegt wird (und nicht durch eine Eingabe oder Berechnung) werden oft mit Großbuchstaben und _ statt Leerzeichen geschrieben.

```
1 let numberValue = 3 + 4;
2 numberValue = 4 + 5;
3
4 const sumOfNumbers = numberValue + 3;
5
6 const DAYS_IN_A_WEEK = 7;
```

Variablen ohne vorherige Deklaration

- Variablen können theoretisch auch ohne vorherige Deklaration verwendet werden
- Es handelt sich dann um sogenannte globale Variablen.
- Die Verwendung solcher ist veraltet und fehleranfällig!
- Es gibt auch Tools (z.B. Linter) um das in der IDE bereits als Fehler markieren zu lassen.



Spezielle Variablenwerte

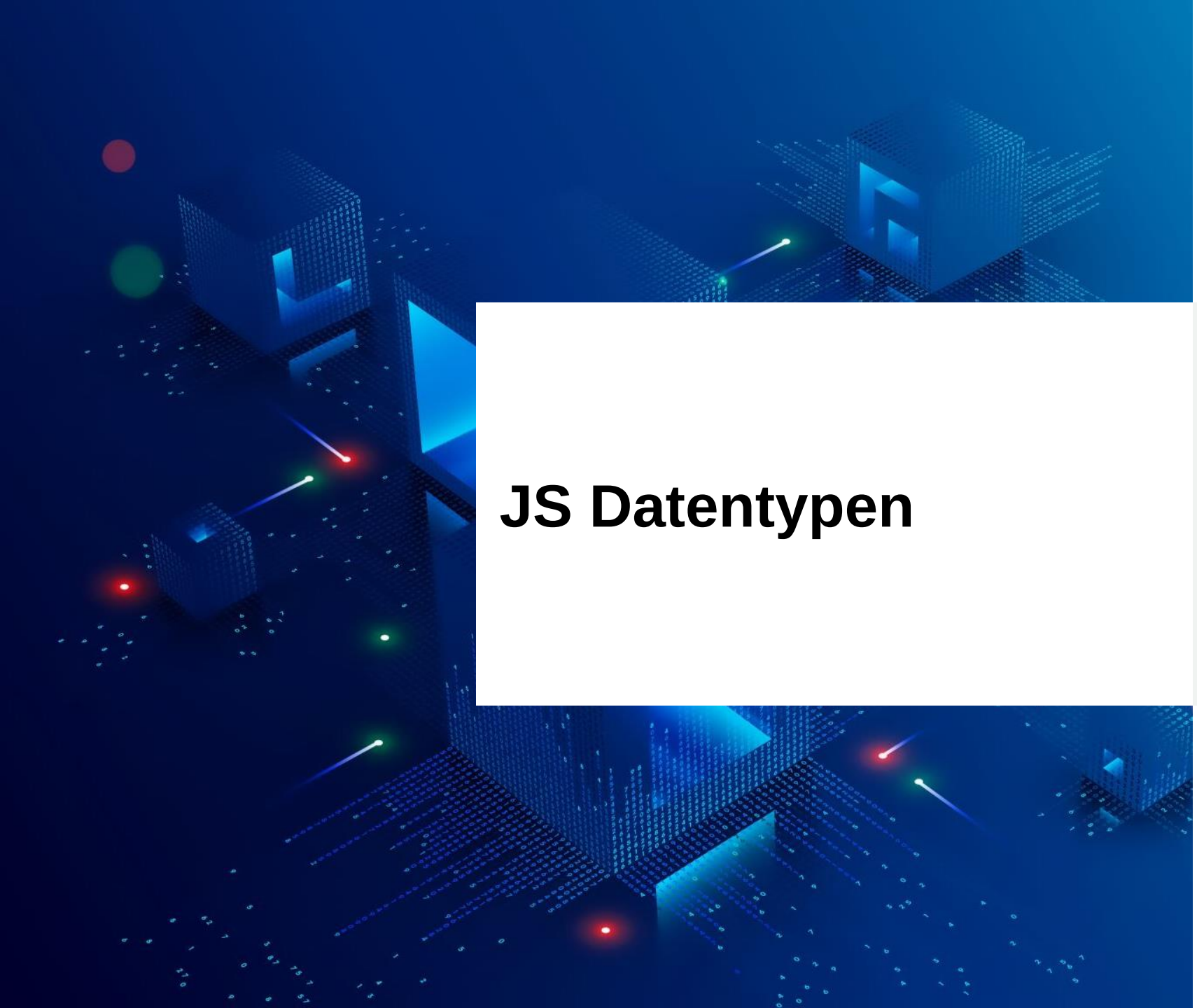
- Es gibt ein paar spezielle Werte, die eine Variable annehmen kann:
- **undefined** - wenn eine Variable definiert, aber kein Wert zugewiesen wurde (eigener Datentyp)
- **null** - repräsentiert das absichtliche Fehlen eines Wertes
- **NaN** - "Not a Number", bei Rechnungen mit ungültigen Werten
- **Infinity** - Unendlich, bei Rechnungen mit zu großen Werten

```
1 let shouldBeUndefined;
2 shouldBeUndefined;
3 let shouldBeNull = null;
4 shouldBeNull;
5 let shouldBeNaN = 12 / "a";
6 shouldBeNaN;
7 let shouldBeInfinity = 12 / 0;
```

Übung zu Variablen

- Ressourcen: 0. Eure Codebeispiele/Dein Name/JavaScript/variables.js
- Schreibe Code mit dem die Werte von a und b vertauscht werden
  - Ziel a = 8 und b = 3
- Es ist nicht erlaubt Zahlen oder Strings einzutippen Bsp: ~~a = „3“~~
- Es ist nicht erlaubt bestehende Code zu verändern

```
2   let a = "3";
3   let b = "8";
4
5   /**********Ändere nicht den Code darüber 👆*******/
6   /💡 Schreibe Code mit dem die Werte für a und b vertauscht werden
7   // Eingabe von Zahlen oder Strings ist nicht erlaubt
8
9
10  /**********Ändere nicht den Code darunter👇*******/
11
12  console.log("a is " + a);
13  console.log("b is " + b);
```



JS Datentypen

Datentyp: Number

- Kurzschreibweisen um den Wert einer Variable zu verrechnen, und anschließend das Ergebnis der selben Variable wieder zuzuweisen: +=, -= , *= , /=, **= , %=.
- Zudem gibt es das **Increment (++)** und **Decrement (--)** das vor oder hinter eine Variable geschrieben wird um ihren Wert um jeweils 1 zu verändern.
- Bei Verwendung vor dem Variablennamen wird die Variable erst verändert und dann verwendet
- Bei Verwendung danach wird die Variable erst verwendet und dann verändert.

```
1 let number = 1;
2 number += 3;
3 number;
4 let otherNumber = number++;
5 otherNumber;
6 number;
7 otherNumber = ++number;
```

Datentyp: String

- Der Datentyp **String** speichert einen beliebigen Text.
- Nutze Anführungszeichen (" oder ') um den Wert als String zu markiereren.
- Öffnendes und schließendes muss identisch sein: " ... " oder ' ... ' , aber nicht " ... '
- Strings können mit dem + Operator aneinandergehängt werden
- Zudem können auch „Template Literals“ verwendet werden
  - String werden zwischen spezielle Anführungszeichen ` ... ` geschrieben
  - Variablen können umgeben von ${ ... } eingefügt werden.

```
1 let firstName = "Hans";
2 let lastName = 'Müller';
3 let badWelcome = "Willkommen " + firstName + " " + lastName;
4 let goodWelcome = `Willkommen ${firstName} ${lastName}`;
5 badFullName;
6 goodFullName;
```

Boolean Datentyp

- Boolean ist ein Datentyp der nur den Wert wahr (true) oder falsch (false) annehmen kann.
- Er wird vor allem bei Verzweigungen im Code beim Vergleich von Werten verwendet.

```
1 const IS_JS_EASY = true;
2 const IS_CPP_EASY = false;
3
4 IS_JS_EASY;
5 IS_CPP_EASY;
```

Datentyp: Object

- **Objekte (Objects)**: In JavaScript sind Objekte komplexe D[illegible]nstrukturen, die Schlüssel-Wert-Paare speichern können und [illegible]schaften als auch Methoden enthalten können.
- [illegible]lle Objekte, die eine geordnete Sammlung von D[illegible]rschiedene Datentypen enthalten und ermöglich[illegible]hand ihres numerischen Index.
- **Dynamische Natur**: Sowohl Objekte [illegible] können zur Laufzeit verändert werden, was Fle[illegible] Manipulation und Verwaltung von Daten bietet.
- **Verwendungszwecke**: Objekte werden häufig zur Darstellung komplexer Entitäten verwendet, während Arrays dazu dienen, geordnete Listen von Elementen zu speichern, auf die über ihre Position im Array zugegriffen werden kann.

Werden wir in kommenden Kapiteln genauer besprechen

```
let myArray = [1, 2, 3, 4, 5];

let myObject = {
    name: "Max",
    age: 30
};
```

Vergleichsoperatoren

- Werte können in JavaScript mit bestimmten Operatoren verglichen werden Für Gleichheit (XNOR) nutzt man ===
- Für Ungleichheit (XOR) nutzt man !==
- Für den und (AND) Vergleich nutzt man &&
- Für den oder (OR) Vergleich nutzt man ||
- Für das Verneinen (NOT) nutzt man ein vorangestelltes !
- Bei Zahlen kann man Größer/-gleich (> / >=) und Kleiner/-gleich (< / <=) prüfen.

```
1 === 3;
1 !== 3;
true && false;
(1 !== 3) || (1 === 3);
!true;
!false;
3 > 3;
3 <= 3;
```

Typensicherheit beim Vergleich

- Wir haben gesehen, dass man für (Un-)Gleichheit die === bzw. !== Zeichen verwendet.
- Es gibt diese Operatoren aber auch ohne das letzte = Zeichen, als == bzw. !=
- Bei der Nutzung von == bzw. != wird nicht sichergestellt, dass beide Variablen auch den selben Typen haben
- Mit Hilfe der Funktion **typeof** kann der Datentyp einer Variable bestimmt werden
- Ressourcen: 7. Javascript/datatypes.js

```
let stringValue = "0";
let numberValue = 0;
stringValue == numberValue;
stringValue === numberValue;
stringValue != numberValue;
stringValue !== numberValue;
typeof(stringValue) //String
typeof(numberValue) //Number
```

String Methoden

String: Length

- **string.length** ist eine Eigenschaft, die die Länge einer Zeichenkette in JavaScript zurückgibt, indem sie die Anzahl der Zeichen in der Zeichenkette zählt.
- Wenn die Variable string den Wert `null` oder `undefined` hat, führt der Aufruf von string.length zu einem Fehler. Man sollte daher vor dem Zugriff auf die Eigenschaft `length` prüfen, ob die Zeichenkette einen gültigen Wert hat.
- string.length ist keine Funktion, sondern eine Eigenschaft. Daher muss man keine Klammern verwenden, um sie aufzurufen. Man greift einfach mit `string.length` darauf zu.

```
// Definiere eine Zeichenkette
let myString = "Hallo, dies ist ein Beispiel.";

// Verwende die length-Eigenschaft, um die Länge der Zeichenkette
let stringLength = myString.length;

// Gib die Länge der Zeichenkette auf der Konsole aus
console.log("Die Länge der Zeichenkette ist: " + stringLength);
```

String: Slice

- Die **slice()** - Funktion wird verwendet, um einen Teil einer Zeichenkette zurückzugeben, wobei die Originalzeichenkette nicht verändert wird.
- **Rückgabewert:** Die `slice()`-Funktion gibt einen neuen Teil der Zeichenkette zurück, der durch den angegebenen Start- und Endindex definiert ist. Die ursprüngliche Zeichenkette bleibt unverändert.
- **Negative Indizes**: Wenn negative Indizes verwendet werden, wird der Startindex vom Ende der Zeichenkette gezählt.
- **Endindex**: Wenn der Endindex ausgelassen wird, wird die `slice()`-Funktion bis zum Ende der Zeichenkette fortgesetzt.

```
// Definiere eine Zeichenkette
let myString = "JavaScript ist eine tolle Sprache.";

// Verwende die slice()-Funktion, um einen Teil der Zeichenkette zurückzugeben
let slicedString = myString.slice(0, 10);

// Gib den ausgeschnittenen Teil der Zeichenkette auf der Konsole aus
console.log("Ausgeschnittene Zeichenkette: " + slicedString);
```

String: toUpperCase & toLowerCase

- Mit diesen Funktionen wird die Groß- oder Kleinschreibung von Zeichenketten geändert, ohne die ursprüngliche Zeichenkette zu verändern.
- **toUpperCase()** - Funktion konvertiert alle Buchstaben in einer Zeichenkette **in Großbuchstaben**.
- **toLowerCase()** - Funktion konvertiert alle Buchstaben in einer Zeichenkette in **Kleinbuchstaben.**
- Beide Funktionen erzeugen neue Zeichenketten und ändern nicht die ursprüngliche Zeichenkette.

```
// Beispiel für toUpperCase():
let myString = "Hallo, Welt!";
let upperCaseString = myString.toUpperCase();
console.log(upperCaseString); // Ausgabe: "HALLO, WELT!"

// Beispiel für toLowerCase():
let anotherString = "DIES IST EIN BEISPIEL";
let lowerCaseString = anotherString.toLowerCase();
console.log(lowerCaseString); // Ausgabe: "dies ist ein beispiel"
```

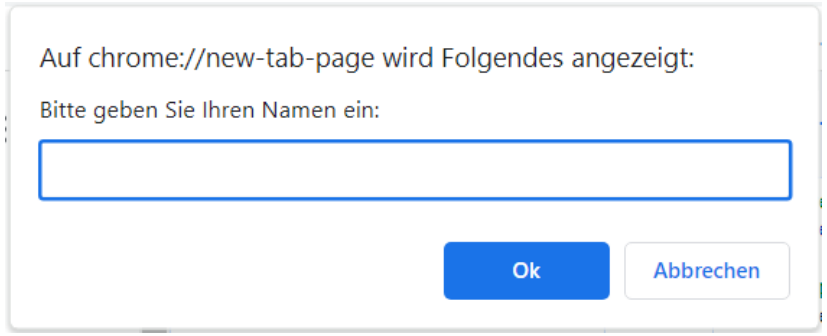
String: indexOf & lastIndexOf

- **indexOf()-**Funktion wird verwendet, um das **erste Auftreten** eines Substrings in einer Zeichenkette zu finden und den Index dieses Auftretens zurückzugeben.
- **lastIndexOf()** wird verwendet, um die Position des **letzten Vorkommens** des gesuchten Substrings zu erhalten
- Falls der Substring nicht gefunden wird, wird -1 zurückgegeben.
- Groß- und Kleinschreibung: Die Funktion unterscheidet zwischen Groß- und Kleinschreibung
- Startposition: Die Funktion akzeptiert optional eine Startposition als zweites Argument, ab der die Suche beginnen soll

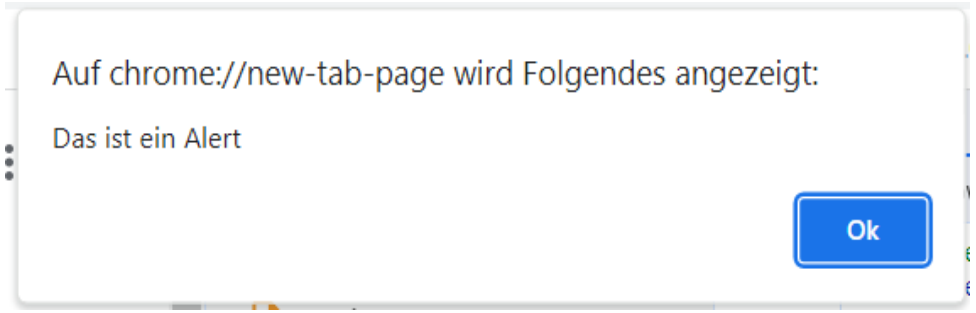
```
let myYodaString = "Coden Ihr wollt lernen junge CodersBay Schüler";

console.log(myYodaString.indexOf("Code")) //  0
console.log(myYodaString.lastIndexOf("Code")) // 29
```

Prompt & Alert



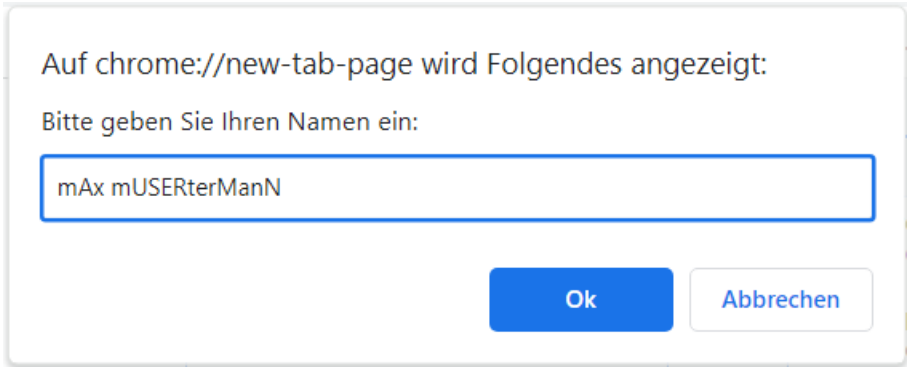
- Die prompt-Funktion wird verwendet, um dem Benutzer ein Eingabefeld anzuzeigen, in dem er Daten eingeben kann.
- Sie nimmt einen optionalen Parameter für die Nachricht, die dem Benutzer angezeigt werden soll, und gibt den eingegebenen Wert als Zeichenkette zurück.
- Beispiel: let name = prompt('Bitte geben Sie Ihren Namen ein:');

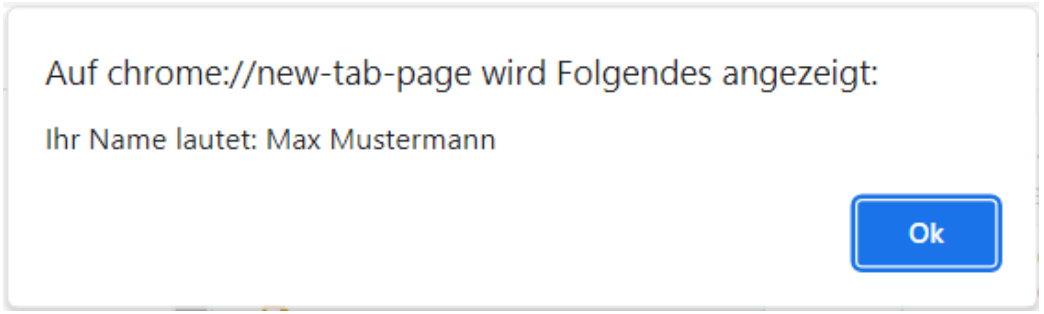


- Die alert-Funktion wird verwendet, um dem Benutzer eine Benachrichtigung anzuzeigen, z. B. eine Erfolgsmeldung oder eine Fehlermeldung.
- Sie nimmt einen Parameter für die Nachricht, die dem Benutzer angezeigt werden soll, und hat kein Rückgabewert.
- Beispiel: alert('Eingabe erfolgreich gespeichert!');

Übung zu Strings

- Schreibe Code um den User über einen Prompt seinen Namen eingeben zu lassen Vor und Nachname Bsp: „max mustermann“
- Manipuliere den String, dass unabhängig von der Art der Eingabe, der Name in einem Alert zurückgegeben wird, wobei die Anfangsbuchstaben von Vor und Nachname groß und der Rest klein geschrieben wird
- Bsp: mAx mUSterMann → Max Mustermann





Zusammenfassung (1)

- **Einleitung:**
  JavaScript ist eine weit verbreitete Skriptsprache, die hauptsächlich für die Entwicklung von interaktiven Webseiten verwendet wird. Sie ermöglicht die dynamische Anpassung von Inhalten und das Verhalten von Webseiten.
- **JS Berechnungen und Datentypen**
  JavaScript ermöglicht Berechnungen mit Zahlen sowie die Manipulation von Strings und Booleans.
  Die Datentypen umfassen Number, String, Boolean und spezielle Datentypen wie Undefined und Null.
- **Number:**
  Der Datentyp "Number" in JavaScript repräsentiert sowohl Ganzzahlen als auch Dezimalzahlen und ermöglicht mathematische Operationen.
- **String:**
  Der Datentyp "String" repräsentiert eine Sequenz von Zeichen und erlaubt die Manipulation von Texten, einschließlich Konkatenation und Ersetzung von Teilstrings.
- **Boolean:**
  Der Datentyp "Boolean" kann entweder "true" oder "false" sein und wird hauptsächlich für bedingte Anweisungen und Schleifen verwendet.

Zusammenfassung (2)

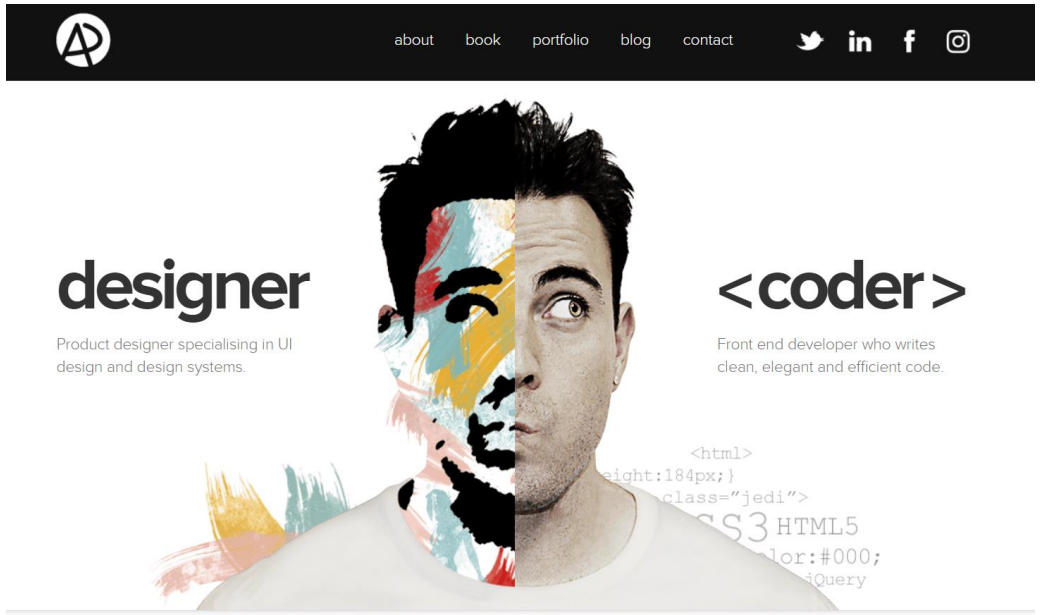
- **Spezielle Datentypen:**
  "Undefined" wird verwendet, wenn eine Variable deklariert, aber nicht initialisiert wurde, während "Null" verwendet wird, um anzuzeigen, dass eine Variable keinen Wert hat.
- **String Methoden:**
  JavaScript bietet verschiedene Methoden zur Manipulation von Strings, darunter "length" zur Ermittlung der Länge, "slice" zur Extrahierung von Teilstrings und "toUpperCase/toLowerCase" zur Änderung der Groß-/Kleinschreibung. "indexOf" und "lastIndexOf" werden verwendet, um die Position eines bestimmten Zeichens oder Teilstrings in einem String zu finden.
- **Promp & Alert:**
  "Prompt" wird verwendet, um den Benutzer um Eingaben zu bitten, während "Alert" verwendet wird, um eine Nachricht anzuzeigen.

JavaScript Übungen als Wiederholung

- https://www.w3schools.com/js/exercise_js.asp?filename=exercise_js_variables1
- JavaScript Exercises
- Variables
- Operators
- Data Types
- Strings
- String Methods

Projekt: Portfolio Website

- Erstellt eure persönliche Portfolio Website für zukünftige Webprojekte
- Holt euch Inspiration von anderen Beispielen aus dem Netz
- Erstellt zunächst ein Konzept bevor ihr mit der Umsetzung startet (Canva, Figma)
- Javascript muss nicht, kann aber verwendet werden
- Zeitplan:
  - ca 10 Stunden
  - verteilt auf die nächsten 5 Einheiten je 2 Stunden

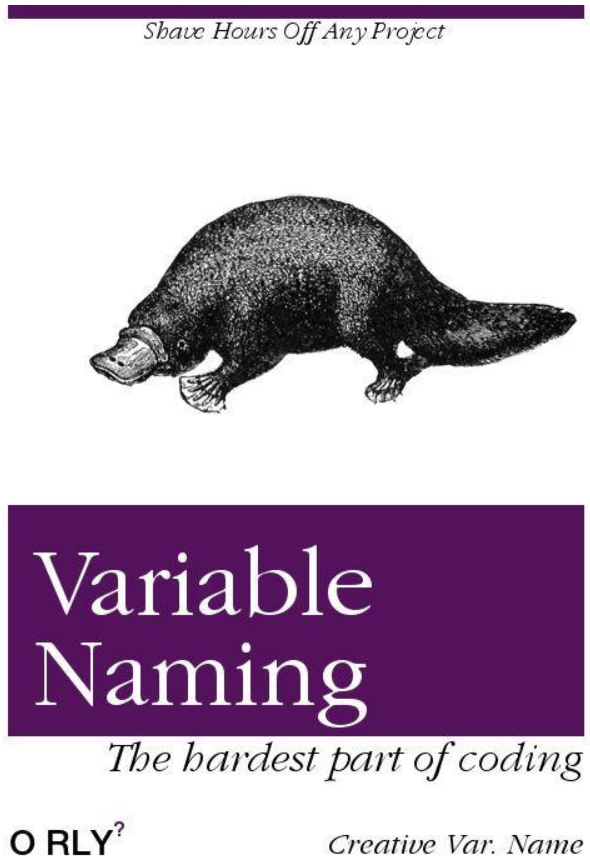


https://www.hostinger.com/tutorials/web-developer-portfolio

Zusatz:
Variablen Benamung

Top 10 der schlechtesten Variablennamen

- ohne Bedeutung
  - data
  - temp
  - a1, a2,..
- Nur ein Buchstabe (außerhalb von Schleifen)
  - i
  - x, ….
- Abkürzungen
  - h // hours
  - m // meters, minutes
- Unaussprechliche Namen
  - txtPwordCfrm
- Falsche Bedeutungen
  - name //Länge des Nachnamen
  - numberOfApples = “10“



Viel Erfolg beim Entwickeln!