Junioraufgabe-1: Sil-ben-tren-nung

Team-ID: 00596
Name: Sophie-RS
23. Oktober 2025
Programmiersprache: TypeScript (NodeJS v22.16.0)

Inhaltsverzeichnis

Problembeschreibung

Lars hat ein Programm erstellt, welches die Silben eines Wortes trennen soll. Dazu hat er eine Reihe an Regeln erstellt, die überprüft werden und entscheiden, ob ein Wort an einer bestimmten Stelle getrennt werden kann.
Es soll nun ein Programm erstellt werden, welches diese Regeln implementiert und die Silbentrennung verbessert.

Lösungsidee

Das Erste, was wir getan haben, war die Regeln von Lars an einem Beispiel anzuwenden, um herauszufinden, welche Fehler diese Regeln haben.
Die aktuellen Regeln lauten wie folgt:

1. Ein Wort kann zwischen zwei Konsonanten getrennt werden.
2. Erster und letzter Buchstabe eines Wortes dürfen nicht abgetrennt werden.
3. Bei drei aufeinanderfolgenden Konsonanten kann nur nach dem ersten getrennt werden.
4. Ein Wort kann nach einem Vokal getrennt werden, falls nicht zwei Konsonanten folgen.

Im *Beispiel02* würden Lars' Regeln den Satz wie folgt trennen:

De-r Ka-pi-tä-n ist eben-fallls Prä-si-dent de-r ör-tlic-he-n Dam-pf-sc-hif-fah-rt-sge-sel-ls-chaft

Was hier direkt auffällt ist, dass viele Silben keine Vokale enthalten (zum Beispiel De-r). Eine Silbe muss jedoch einen Vokal beinhalten, da der Vokal in einer Silbe der „Klangträger" ist. Ohne einen Vokal könnte man eine Silbe nicht aussprechen.
Eine Silbe benötigt also mindestens einen der folgenden Buchstaben:

a, e, i, o, u, ö, ä, ü

Die Regel *„Ein Wort darf nur getrennt werden, wenn beide Silben, die durch das Trennen entstehen, einen Vokal enthalten."* wird nun zu der Regelliste hinzugefügt. Da die Regeln nach ihrer Wichtigkeit von unten nach oben sortiert sind, platzieren wir diese Regel am Ende der Liste, damit sie die höchste Priorität hat.

Was bei der fehlerhaften Trennung ebenfalls auffällt: Buchstabenpaare wie *„sch"* werden in der Mitte getrennt. Das ist jedoch nicht erlaubt, da diese Paare als ein einziger Laut ausgesprochen werden.
Unser nächster Gedanke war daher, eine Regel zu erstellen, die das Trennen solcher Buchstabenpaare verhindert.
Doch was genau sind diese Buchstabenpaare?
Man unterscheidet bei den Paaren zwischen Diphthongs, Digraphen und Trigraphen:
Diphthongs bestehen aus zwei Vokalen.
au, ei, ai, eu, äu, oi, ui, ee, aa, ie
Digraphen bestehen aus zwei Konsonanten (also Buchstaben, die keine Vokale sind):
ck, ch, ph, th, qu

Trigraphen bestehen aus drei Konsonanten. Es gibt hier nur das eine Paar ***sch***.

Diese drei Listen kann man nun zu einer Liste zusammenführen. Dies ist dann die Liste der Buchstabenpaare, die nicht getrennt werden dürfen.
Die Regel, die nun vor die Vokal-Regel gesetzt wird, lautet: *„Ein Wort darf nur getrennt werden, wenn dabei kein Buchstabenpaar getrennt wird.“*

Mit den beiden Regeln wird das *Beispiel02* nun wie folgt getrennt:

Der Ka-pi-tän ist eben-fallls Prä-si-dent der örtlichen Dampf-schif-fah-rtsge-sel-lschaft

Diese Trennung ist zwar schon deutlich besser, allerdings treten weiterhin einige Fehler auf.
Eine grundlegende Eigenschaft einer Silbe ist, dass sie aussprechbar ist. Versucht man jedoch, die gebildeten Silben nun auszusprechen, merkt man schnell, dass das bei manchen nicht möglich ist, wie zum Beispiel bei „rtsge“ in „Dampfschiffahrtsgesellschaft“.
Dieses Problem lässt sich mit sogenannten Anlauten lösen. Das sind Buchstaben oder Buchstabenpaare, mit denen eine Silbe beginnen muss. Ein solcher Anlaut steht zwischen dem Anfang einer Silbe und dem ersten Vokal. Ein Anlaut ist Valide, wenn er sich in der folgenden Liste befindet:

sch, bl, br, dr, fl, fr, gl, gr, gn, kl, kr, pl, pr, pf, tr, st, sp, ch, ph, th, qu, a, e, i, o, u, ä, ö, ü, y, b, d, f, g, h, j, k, l, m, n, p, r, s, t, v, w, z, ß

Eine Silbe muss also mit einem dieser Buchstabenpaare beginnen. Diese Regel wird nun vor der Buchstabenpaar-Regel platziert und lautet: *„Ein Wort darf nur getrennt werden, wenn die beim Trennen entstehende Silbe mit einem Anlaut beginnt.“*

Nun sieht die aktuelle Regelliste wie folgt aus:

1. Ein Wort kann zwischen zwei Konsonanten getrennt werden.
2. Erster und letzter Buchstabe eines Wortes dürfen nicht abgetrennt werden.
3. Bei drei aufeinanderfolgenden Konsonanten kann nur nach dem ersten getrennt werden.

4. Ein Wort kann nach einem Vokal getrennt werden, falls nicht zwei Konsonanten folgen.
5. Ein Wort darf nur getrennt werden, wenn die beim Trennen entstehende Silbe mit einem Anlaut beginnt.
6. Ein Wort darf nur getrennt werden, wenn kein Buchstabenpaar dabei getrennt wird.
7. Ein Wort darf nur getrennt werden, wenn beide Silben, die durch das Trennen entstehen, einen Vokal beinhalten.

Mit diesen Regeln wird Beispiel02 nun korrekt getrennt.

Der Ka-pi-tän ist e-ben-fallls Prä-si-dent der ör-tli-chen Dampf-schif-fahrts-ge-sell-schaft

Im Beispiel08 werden jedoch noch einige Wörter nicht korrekt oder garnicht getrennt wie zum Beispiel „Betre-ten“, „Himmlische“ oder „getheilt“. Was bei dieser Trennung auffällt ist, dass keine falschen Trennungen entstehen, sondern dass Trennungen fehlen. Um diese fehlenden Trennungen zu beheben, ergänzen wir die neue Regel *„Trenne das Wort, wenn es noch nicht getrennt wurde.“* Diese Regel wird nun an den Anfang der Regelliste gesetzt, womit sie die geringste Priorität hat.
Durch diese Regel kann zwar nun ein Konflikt mit fast jeder anderen Regel entstehen, doch da sie ganz oben steht, wird sie nur dann angewendet, wenn keine andere Regel greift.
Die Regeln funktionieren nun so, das sie an jeder Stelle im Wort prüfen, ob dort getrennt werden kann. Wenn keine Regel zutrifft die dies verbietet oder tut, wird durch die neue Regel immer getrennt. Sollte also keine untere Regel (2-8) zutreffen, wird getrennt. Aus diesem Grund kann Regel-2 entfernt werden. Regel-2 besagte, dass zwischen zwei Konsonanten getrennt werden darf. Da sie jedoch weit oben in der Liste steht und somit eine niedrige Priorität hat, ist sie überflüssig. Denn wenn gerade nicht zwischen zwei Konsonanten getrennt wird und alle anderen Regeln nicht zutreffen, wird durch die oberste Regel trotzdem getrennt. Es ist also nicht mehr nötig zu prüfen, ob zwischen zwei Konsonanten getrennt werden kann, da in beiden Fällen ohnehin eine Trennung erfolgt. Die ursprüngliche Regel von Lars *„Ein Wort kann zwischen zwei Konsonanten getrennt werden“* wird daher entfernt.

Die endgültige Regelliste lautet nun:

1. Trenne das Wort, wenn es noch nicht getrennt wurde.
2. Erster und letzter Buchstabe eines Wortes dürfen nicht abgetrennt werden.
3. Bei drei aufeinanderfolgenden Konsonanten wird nach dem ersten getrennt.
4. Ein Wort darf nach einem Vokal getrennt werden, wenn nicht zwei Konsonanten folgen.
5. Ein Wort darf nur getrennt werden, wenn die beim Trennen entstehende Silbe mit einem Anlaut beginnt.
6. Ein Wort darf nur getrennt werden, wenn kein Buchstabenpaar dabei getrennt wird.
7. Ein Wort darf nur getrennt werden, wenn beide Silben, die durch das Trennen entstehen, einen Vokal beinhalten.

Sonderfälle beim Trennen mit diesen Regeln sind die Wörter „Karpfen“, „Radio“, „ebenfalls“ und „Angle“. Diese Wörter werden korrekt nach den Sprechsilben getrennt, aber nicht grammatikalisch korrekt. Da wir uns aber dafür entschieden habe, nach Sprechsilben zu trennen, ist das nicht problematisch.

Außerdem werden englische Wort wie „Hobbyte“ aus Beispiel07 mit diesen Regeln falsch zu „Hobby-te“ getrennt. Dies liegt daran, dass wir im Englischen das „y“ oft wie ein „i“ (also einen Vokal) aussprechen und das „e“ hier am Ende stumm ist. Da englische Wörter eine komplett andere Silbentrennung haben als deutsche, haben wir uns entschieden, englische Wörter nicht mit diesen Regeln zu trennen. Um „Hobbyte“ korrekt zu trennen, bräuchte es zusätzliche Regeln, die englische Wörter erkennen und stumme „e“ berücksichtigen. Solche Regeln wären zu kompliziert und es würde viele andere Probleme entstehen.

Mit unseren Regeln können wir nun viele Wörter korrekt nach den Sprechsilben trennen, jedoch gibt es ein paar Ausnahmen. Wörter mit „pf“ im Wort, bei denen „pf“ getrennt wird, werden ebenfalls nicht richtig behandelt. So trennt das Programm das Wort „Impfen“ im Eigenen-Beispiel01 zu „Im-pfen“, obwohl es eigentlich „imp-fen“ heißen müsste. Dieses Problem entsteht, weil „pf“ nur

in bestimmten Fällen ein valider Anlaut ist. Da es aber seltener ist, dass „pf“ nicht als Anlaut fungiert, haben wir uns entschieden, „pf“ in allen Fällen als Anlaut zu behandeln. Die Fälle, in denen „pf“ kein Anlaut ist, sind komplex und nicht ohne Weiteres durch einfache Regeln abbildbar. Durch diesen Kompromiss trennt das Programm Wörter wie „Impfen“, „Tropfen“ oder „Schnupfen“ falsch.

Umsetzung

Wir haben das gesamte Programm in TypeScript geschrieben und mit NodeJS, Version 22.16.0, ausgeführt. Sowohl der TypeScript- als auch der JavaScript-Code sind in der ZIP-Datei enthalten. In dieser Dokumentation werden wir jedoch nur den Code aus der TypeScript-Version erläutern, da die JavaScript-Version auf dem TypeScript-Code basiert und daraus kompiliert wurde.
Wir haben uns dazu entschieden, das Programm konsolenbasiert zu programmieren. Das bedeutet, dass jegliche Ein- und Ausgabe innerhalb der Konsole erfolgt.
Um der Konsolenausgabe einen schöneren Stil zu geben, haben wir das Modul *consoleLogger.ts* in das Programm importiert. Dabei handelt es sich um eine Sammlung selbst geschriebener Funktionen, die das farbige Schreiben in der Konsole mithilfe von Color-Codes vereinfachen. Sie funktionieren so, dass man bestimmte Parameter an eine Funktion übergeben kann, aus denen anschließend ein String erstellt wird, der die entsprechenden Color-Codes enthält.
Da dies nichts mehr mit der eigentlichen Aufgabe zu tun hat, werden wir hier weder auf die genaue Funktionsweise noch auf die konkrete Nutzung in diesem Programm eingehen.
Wenn wir in dieser Dokumentation von „in der Konsole ausgeben“ schreiben, ist damit ein Text gemeint, der von einer dieser Funktionen erzeugt wurde.

Um die Konsoleneingabe zu realisieren, haben wir das npm-Package *readline* in das Programm importiert. Von diesem Package nutzen wir die Funktion *createInterface()*, die ein Objekt zurückgibt, über das die Methode *question(question: string, callback: Function)* aufgerufen werden kann, um den Benutzer nach einer Eingabe zu fragen.
Diese Funktion wird zu Beginn direkt benutzt, um den Nutzer nach einem Text zu fragen, welcher in Sprechsilben getrennt werden soll.

Nachdem der Nutzer den Text eingegeben hat, gibt das Programm ihn wieder aus, nun jedoch mit „|“-Symbolen im Text, die die Trennstellen markieren.

```
// Programm ausführen
console.log(header);

rl.question(question, (answer) => {
  console.log(headerErgebnis);

  console.log(createColorText("WHITE", "REGULAR", true, trennSilben(answer)));

  console.log(ende);

  rl.close();
});
```

Diese Trennstellen werden durch die Funktion *trennSilben(text: string): string* gesetzt, welche wie folgt funktioniert:

Zuerst erstellt sie mithilfe der String-Funktion *split(separator: string): string[]* ein Array, bei dem der String an jedem Leerzeichen geteilt wird. So ist nun jedes Wort einzeln im Array enthalten.

```
export default function trenneSilben(text: string): string {
  return text.split(" ").map(splitWort).join(" ");
}
```

Anschließend wird über dieses Array mithilfe der *map-Funktion* iteriert. Diese funktioniert so, dass man ihr eine Funktion übergibt, die für jedes Element des Arrays aufgerufen wird. Der Rückgabewert dieser Funktion wird dann in einem neuen Array gespeichert. Am Ende gibt *map(...)* also ein neues Array mit all diesen Rückgabewerten zurück.
In diesem Fall übergibt das Programm die Funktion *splitWort(wort: string): string*, welche das übergebene Wort getrennt zurückgibt.

Diese Funktion teilt das Wort in Sprechsilben, indem sie zunächst ein Array erstellt, das nur die einzelnen Buchstaben des Wortes enthält.
Dafür wird das Wort zuerst in Kleinbuchstaben umgewandelt. Das hat den Vorteil, dass die spätere Überprüfung einfacher wird, da nicht zwischen Groß- und Kleinschreibung unterschieden werden muss, und so jedes Vokal- oder Buchstabenpaar nur einmal in der Vergleichsliste existieren muss.

```
function splitWort(wort: string): string {
  const wortArray = wort
    .toLowerCase()
    .replaceAll(/[^a-zäöüß]/g, "")
    .split("");

  const positionSilbe: number[] = [];
```

Nachdem das Wort nun vollständig in Kleinbuchstaben vorliegt, werden alle Zeichen entfernt, die nicht zum Alphabet gehören oder Sonderbuchstaben wie „ö“, „ä“, „ü“ oder „ß“ sind. Dies geschieht mit der Funktion replaceAll(regex: Regex, replacement: string): string, welche alle Zeichen im String, die vom angegebenen regulären Ausdruck erfasst werden, durch einen leeren String ersetzt werden.

Zum Schluss wird das bereinigte Wort mit der Funktion *split(separator: string): string[]* an jedem Zeichen getrennt, sodass jedes Zeichen nun als einzelnes Element im Array vorliegt.
Zusätzlich zu diesem Wort-Array wird noch ein Array erstellt, welches später die Positionen der Silben im Wort beinhaltet.

Nun beginnt das Programm, über das Wort-Array zu iterieren. Es startet dabei bei Position 0 und läuft bis Position *n - 1*, wobei *n* die Länge des Wortes beschreibt. Doch was bedeutet „Position 0“ genau? Wenn das Programm an Position 0 ist, überprüft es, ob zwischen dem ersten und zweiten Buchstaben getrennt werden kann.
Die Variable *i* steht für die mögliche Trennstelle im Wort und ist der Index der Schleife, die über das Wort-Array iteriert. Sie beschreibt also die Position zwischen dem Buchstaben an Stelle *i* und dem Buchstaben an Stelle *i + 1*.

Nun werden in dieser *for*-Schleife die Regeln überprüft. Die Regeln sind im Grunde einfach Bedingungen, die nacheinander abgefragt werden. Da die unterste Regel die höchste Priorität besitzt, wird mit dieser begonnen.
Die erste Regel, die also überprüft wird, ist Regel-7:

„Ein Wort kann nur getrennt werden, wenn beide Silben, die durch das Trennen entstehen, einen Vokal beinhalten.“

Es muss also geprüft werden, ob links und rechts neben der möglichen Trennstelle jeweils ein Vokal vorhanden ist. Dies geschieht mit der Funktion *besitzenSilbenVokale(position: number): boolean*. Diese Funktion überprüft, ob bestimmte Teilbereiche des Wort-Arrays einen Vokal enthalten. Dabei werden zwei Teilstücke gebildet: das Teilstück von der letzten Trennposition bis zur aktuellen Position und das Teilstück von der aktuellen Position bis zum Ende des Arrays.

```
function besitzenSilbenVokale(position: number): boolean {
  return (
    wortArray
      .slice(positionSilbe.at(-1) + 1, position + 1)
      .some((b) => vokale.includes(b.toLowerCase())) &&
    wortArray.slice(position + 1).some((b) => vokale.includes(b.toLowerCase()))
  );
}
```

Das Array wird dazu mit der Funktion *slice(...): string[]* geteilt. Anschließend wird mit der Funktion *some(...): boolean* überprüft, ob ein Vokal aus der Vokalliste in einem dieser Teilstücke vorkommt.
Sollte nun kein Vokal vorhanden sein, wird der entsprechende if-Block der grade überprüften Regel ausgeführt. Damit wird in der *for*-Schleife die aktuelle Iteration übersprungen, und das Programm fährt mit der nächsten Position

fort.
Sollten nun beide Teilstücke rechts und links neben der Trennposition einen Vokal beinhalten, wird mit der nächsten Regel fortgefahren, Regel-6:

„Ein Wort darf nur getrennt werden, wenn kein Buchstabenpaar dabei getrennt wird.“

Diese Regel wird mit der Funktion *istPaar(position: number): boolean* überprüft. Diese Funktion prüft, ob sich an der aktuellen Position ein Buchstabenpaar befindet, das in der Paar-Liste definiert ist. Sollte an dieser Position tatsächlich ein solches Paar vorkommen, wird zusätzlich überprüft, ob dieses Paar mit „s“ beginnt, also ob es sich um „sch“ handelt. In diesem Fall kann die Schleife, die durch jede mögliche Position läuft, eine Position überspringen, da innerhalb des „sch“ keine weiteren Überprüfungen mehr notwendig sind. Nachdem dies überprüft wurde und, falls das Paar tatsächlich ein „sch“ war, und eine Position übersprungen wurde, bricht das Programm die aktuelle Iteration ab und setzt die Schleife mit der nächsten Position fort.

```
function istPaar(pos: number): boolean {
  if (pos > wortArray.length - 2) return false;

  if (
    paare.some((value) => value.substring(0, 2) === wortArray.slice(pos, pos + 2).join(""))
  ) {
    if (wortArray.slice(pos, pos + 2000).join("") !== "sc")
      return true;

    if (wortArray.slice(pos, pos + 3).join("") === "sch")
      return true;
  } else return false;
}
```

Die Überprüfung, ob sich die aktuelle Position in einem Paar befindet, funktioniert indem zuerst überprüft wird, ob die Position am Ende des Arrays liegt. Ist das der Fall, kann sie sich in keinem Paar mehr befinden, da ein Paar immer aus mindestens zwei Buchstaben bestehen muss.
Befindet sich die Position nicht am Ende des Wortes wird geprüft, ob eines der bekannten Paare mit den beiden Buchstaben beginnt, die an dieser Position stehen.
Trifft das zu wird überprüft, ob es sich dabei nicht um „sc“ handelt. Wenn es kein „sc“ ist, gibt die Funktion *true* zurück, da sich an dieser Stelle ein Buchstabenpaar befindet.
Handelt es sich jedoch um „sc“, wird überprüft, ob auf „sc“ noch ein „h“ folgt. Sollte das zutreffen, befindet sich die Position im Paar „sch“, und die Funktion gibt ebenfalls *true* zurück.
Wenn das Programm kein Paar findet, das mit den beiden Buchstaben an dieser Position beginnt, gibt die Funktion *false* zurück.

Nachdem nun sicher gestellt wurde, dass sich die aktuelle Position nicht in einem Buchstabenpaar befindet, wird als nächstes die Regel

„Ein Wort darf nur getrennt werden, wenn die beim Trennen entstehende Silbe mit einem Anlaut beginnt.“

überprüft. Diese Überprüfung erfolgt mit der Funktion *startetMitAnlaut(position: number): boolean*. Gibt diese Funktion *false* zurück, beginnt die Silbe, die beim Trennen entstünde, nicht mit einem Anlaut, das Programm bricht also die aktuelle Iteration ab und setzt mit der nächsten Position fort.
Die Prüfung, ob die entstehende Silbe mit einem Anlaut beginnen würde, funktioniert so:

Zuerst wird geprüft, ob sich direkt neben der möglichen Trennstelle ein Vokal befindet. Diese Überprüfung erfolgt mit der Funktion *istVokal(position: number, additionToPos: number): boolean*. Sie prüft, ob sich an der angegebenen Position, gegebenenfalls mit einer Addition zur Position, ein Vokal befindet. Wird kein zusätzlicher Wert übergeben, ist *additionToPos* standardmäßig 0. Die Funktion überprüft dann, ob an der berechneten Position ein Buchstabe steht, der in der Vokalliste enthalten ist. Ist die berechnete Position größer als die Länge des Wortes, wird *false* zurückgegeben. Sollte nun der Buchstabe in der Vokalliste sein, also ein Vokal sein, wird *true* zurück-geben, wenn nicht *false.*

```
function istVokal(position: number, additionToPos: number = 0): boolean {
  if (!wortArray[position + additionToPos]) return false;

  return vokale.includes(wortArray[position + additionToPos].toLowerCase());
}
```

Ist nun neben der Trennstelle ein Vokal, wird sofort *true* zurückgegeben, denn ein Vokal ist ein valider Anlaut. Beginnt die mögliche Silbe nicht mit einem Vokal, wird die Position des nächsten Vokals ermittelt, indem eine *while*-Schleife so lange läuft, bis sie entweder das Ende des Arrays erreicht oder einen Vokal findet. Mit dieser Position kann nun geprüft werden, ob sich zwischen der Trennstelle und diesem Vokal ein valider Anlaut befindet.
Dazu wird zunächst geprüft, ob der Abstand zwischen der möglichen Trennstelle und dem Vokal größer oder gleich 3 ist. In diesem Fall liegt dort ein einfacher Anlaut (ein einzelnes Anlautmuster) vor. So wird nun überprüft, ob der *Substring* zwischen Trennstelle und Vokal in der Anlautliste steht. Ist der Abstand größer als 3, kann der Anlaut aus „sch“ + Einzelanlaut bestehen (z. B. im Eigenen-Beispiel02 hat „Schranke“ in der ersten Silbe den Anlaut „schr“).

Dann wird geprüft, ob die Silbe mit „sch“ beginnt und danach ein Einzelanlaut folgt. Konkret bedeutet das: Es wird geprüft, ob der Abstand genau 4 beträgt. Ist er nicht 4, wird *false* zurückgegeben, da es keinen validen Anlaut mit mehr als vier Buchstaben gibt. Beträgt der Abstand 4, wird der *Substring* von Position + 1 bis Position + 4 genommen und geprüft, ob er „sch“ entspricht. Trifft das zu, wird der darauffolgende Buchstabe daraufhin überprüft, ob er ein valider Einzelanlaut ist (also ein Anlaut, der nur aus einem Buchstaben besteht). Ist auch das der Fall, gibt die Funktion *true* zurück. So sind alle Anlaut-Möglichkeiten nun geprüft und die mögliche Silbe beginnt mit einem validen Anlaut.

```
function startetMitAnlaut(position: number): boolean {
  if (istVokal(position + 1)) return true;

  let nextVokal = position + 1;

  while (!istVokal(nextVokal) && position < wortArray.length - 1) nextVokal++;

  const abstandZuVokal = nextVokal - 1 - position;

  if (abstandZuVokal <= 3 && abstandZuVokal > 0) {
    return anlaute.flat(2).some((v) => v === wortArray.slice(position + 1, nextVokal).join(""));
  } else if (abstandZuVokal === 4) {
    return (
      wortArray.slice(position + 1, position + 4).join("") === anlaute[0][0] &&
      anlaute[2].some((v) => v === wortArray[position + 4])
    );
  }
  return false;
}
```

Nun führt das Programm mit der nächsten Regel fort, welche die folgende ist:

„Ein Wort darf nach einem Vokal getrennt werden, wenn nicht zwei Konsonanten folgen.“

Diese Regel ist die erste, die eine mögliche Trennung tatsächlich vornimmt. Wenn auf einen Vokal nicht zwei Konsonanten folgen, wird die aktuelle Position dem Array *positionSilbe* hinzugefügt. Geprüft wird dies indem zuerst festgestellt wird, ob sich an der aktuellen Position ein Vokal befindet. Ist das der Fall, wird zusätzlich geprüft, ob dieser Vokal der vorletzte Buchstabe des Wortes ist. Dann folgt nämlich nur noch ein Buchstabe, es können also keine zwei Konsonanten mehr folgen. Liegt die Position nicht am Ende des Arrays wird geprüft, ob einer der beiden Buchstaben, die nach dieser Position kommen, ein Vokal ist. Trifft das zu, folgen nicht zwei Konsonanten, und das Programm darf trennen.

Sollte das Programm nun keine Trennung durchgeführt haben, wird mit der nächsten Regel fortgefahren:

„Bei drei aufeinanderfolgenden Konsonanten wird nach dem ersten getrennt.“

Diese Regel führt also wieder eine Trennung durch, jedoch nur dann, wenn tatsächlich drei Konsonanten aufeinander folgen.

Um diese Bedingung zu prüfen, wird zunächst überprüft, ob sich die aktuelle Position nicht innerhalb der letzten drei Buchstaben des Wortes befindet. Ist das der Fall, können dort keine drei Buchstaben und somit auch keine drei Konsonanten mehr folgen.
Liegt die Position hingegen nicht am Ende des Wortes, wird überprüft, ob an der aktuellen Position kein Vokal steht und ob auch an den Positionen +1 und +2 keine Vokale stehen. Wenn an allen drei Positionen kein Vokal vorhanden ist, kann eine Trennung nach der aktuellen Position vorgenommen werden, und so wird die aktuelle Position dem Array *positionSilbe* hinzugefügt.

Wird auch hier nicht getrennt, fährt das Programm mit der nächsten Regel fort:

„Erster und letzter Buchstabe eines Wortes dürfen nicht abgetrennt werden.“

Diese Regel zu überprüfen ist recht einfach. Sie besagt, dass zwischen dem ersten und zweiten sowie dem vorletzten und letzten Buchstaben keine Trennung stattfinden darf.
Das Programm prüft also, ob die aktuelle Position entweder 0 oder größer bzw. gleich der Wortlänge minus 2 ist.
Der Wert -2 wird benötigt, weil der letzte Buchstabe abgetrennt werden würde, wenn sich die Position am vorletzten Buchstaben befindet, da die Position immer die Trennstelle zwischen dem aktuellen Buchstaben und dem darauffolgenden (+1) beschreibt.
Befindet sich die Position also am Anfang oder Ende des Wortes, wird die aktuelle Iteration abgebrochen und mit der nächsten Position fortgesetzt.

Sollte nun auch diese Regel nicht zutreffen wird mit der letzten Regel fortgefahren:

„Trenne das Wort, wenn es noch nicht getrennt wurde.“

Diese Regel besagt ganz simpel, dass das Wort an dieser Stelle nun getrennt wird. Also wird dies auch getan.

Wenn das Programm nun für jede mögliche Trennstelle alle Regeln überprüft hat, muss das Wort mit den ermittelten Trennstellen wieder zusammengesetzt werden.
Dafür wird eine Ergebnis-Variable erstellt, die am Ende das fertig getrennte Wort enthalten soll. Anschließend wird eine *for*-Schleife gestartet, die über das ursprüngliche Wort iteriert. Der Index *i* dieser Schleife entspricht der aktuellen Position im Originalwort und hat eine maximale Größe, die der Länge des

```
// Füge das Wort mit Trennstellen wieder zusammen
const trennStrich = "\x1b[1;30m|\x1b[0;37m";
let ergebnis = "";
for (let i = 0, j = 0; i < wort.length; i++) {
  if (wort[i].toLowerCase() === wortArray[j]) {
    ergebnis = ergebnis + wort[i];

    if (positionSilbe.some((value) => value === j)) ergebnis += trennStrich;
    j++;
  } else {
    ergebnis += wort[i];
  }
}

return ergebnis;
```

Wortes entspricht.
Zusätzlich zu diesem Index wird der Index *j* verwendet, der die Position im Wort-Array darstellt.
Zuerst überprüft das Programm nun, ob der Buchstabe an der Position *i* im Originalwort derselbe ist wie der Buchstabe an der Position *j* im Wort-Array, wobei die Groß- und Kleinschreibung ignoriert wird. Ist das der Fall, wird das Zeichen an der Position *i* im Originalwort an die Ergebnisvariable angehängt.
Anschließend prüft das Programm, ob das Array *positionSilbe* einen Wert enthält, der dem aktuellen *j* entspricht. Ist das so, wird zusätzlich ein Trennstrich („|“) an das Ergebnis angehängt, da dies bedeutet, dass an dieser Stelle laut den Regeln getrennt werden soll. Danach wird *j* um 1 erhöht.
Sollte der Buchstabe an der Position *i* im Originalwort nicht dem an der Position *j* im Wort-Array entsprechen, wird das Zeichen an der Position *i* im Originalwort trotzdem an das Ergebnis angehängt, allerdings ohne zu prüfen, ob dort eine Trennstelle ist, und ohne *j* zu erhöhen.
Das liegt daran, dass dieses Zeichen nicht im Wort-Array enthalten ist, es handelt sich dabei wahrscheinlich um ein Sonderzeichen wie ein Komma oder einen Punkt und muss daher übersprungen werden.
Das Ergebnis wird dann von der Funktion zurückgeben.

Nachdem nun alle Wörter getrennt wurden, werden sie mithilfe der Array-Funktion *join(separator: string): string* wieder zu einem einzigen String zusammengesetzt, wobei zwischen jedem Wort ein Leerzeichen eingefügt wird. Dieser zusammengesetzte String wird anschließend von der Funktion *trennSilben(text: string): string* zurückgegeben und vom Programm ausgegeben.

Werkzeuge

- **Visual Studio Code:** Wir haben das gesamte Programm mit der IDE Visual Studio Code geschrieben. Diese Umgebung bietet Code-Completion und eine integrierte TypeScript-Unterstützung, durch die die automatische Codevervollständigung ermöglicht wird. Jeglicher Code,

der in dieser Dokumentation dargestellt ist, wurde in Visual Studio Code fotografiert.

- **NPM:** NPM (Node Package Manager) ist ein Paketmanager, mit dem das Programm zusätzliche Packages nutzen kann. In diesem Projekt sind das Readline-Package und die TypeScript-Declarations für das fs-Modul von JavaScript installiert. NPM vereinfacht außerdem die Ausführung des Programms, da damit Skripte erstellt werden können, die mehrere Befehle nacheinander ausführen.
- **ChatGPT (GPT-5):** ChatGPT wurde hauptsächlich zur Korrektur von Rechtschreibung und Satzbau verwendet, wobei wir versucht haben, unseren eigenen Schreibstil so weit wie möglich beizubehalten
- **Internet:** Das Internet wurde genutzt, um bestimmte Funktionen in Javascript zu recherchieren.
- **Apple Pages:** Apple Pages wurde zum Schreiben dieser Dokumentation verwendet. Dabei wurde keine integrierte KI-Funktion eingesetzt.
- **Google-Docs:** Google docs wurde benutzt, um zusammen an der Dokumentation zu arbeiten.
- **Familie:** Diese Dokumentation haben Teile unserer Familien gelesen, um Satzbau und Aufgaben korrekt zu überprüfen.

Beispiele

Beispiel01: Mein Name ist Lars und ich esse schrecklich gerne Sauerkraut.

```
-Jugendwettbewerb-2025-43-Runde-3- Junioraufgabe-2 Sophie-RS

Gebe den Text ein, welcher getrennt werden soll: Mein Name ist Lars und ich esse schrecklich gerne Sauerkraut.

---Ergebnis-----------------------------
Mein Na|me ist Lars und ich es|se schreck|lich ger|ne Sau|er|kraut.
----------------------------------------
```

Beispiel02: Der Kapitän ist ebenfalls Präsident der örtlichen Dampfschifffahrtsgesellschaft.

```
-Jugendwettbewerb-2025-43-Runde-3- Junioraufgabe-2 Sophie-RS

Gebe den Text ein, welcher getrennt werden soll: Der Kapitän ist ebenfalls Präsident der örtlichen Dampfschifffahrtsgesells
chaft.

---Ergebnis-----------------------------
Der Ka|pi|tän ist e|ben|falls Prä|si|dent der ört|li|chen Dampf|schiff|fahrts|ge|sell|schaft.
----------------------------------------
```

Beispiel03: Es ist arschkalt!

```
-Jugendwettbewerb-2025-43-Runde-3- Junioraufgabe-2 Sophie-RS

Gebe den Text ein, welcher getrennt werden soll: Es ist arschkalt!

---Ergebnis-----------------------------
Es ist ar|schkalt!
----------------------------------------
```

Beispiel04: Ist das Audiosignal im Radio schlecht?

```
-Jugendwettbewerb-2025-43-Runde-3- Junioraufgabe-2 Sophie-RS

Gebe den Text ein, welcher getrennt werden soll: Ist das Audiosignal im Radio schlecht?

---Ergebnis-----------------------------
Ist das Au|di|o|si|gnal im Ra|di|o schlecht?
----------------------------------------
```

Beispiel05: Sein Vater ist Bauer und erntet Mais.

```
-Jugendwettbewerb-2025-43-Runde-3- Junioraufgabe-2 Sophie-RS

Gebe den Text ein, welcher getrennt werden soll: Sein Vater ist Bauer und erntet Mais.

---Ergebnis-----------------------------
Sein Va|ter ist Bau|er und ern|tet Mais.
----------------------------------------
```

Beispiel06: Ich angle Karpfen.

```
-Jugendwettbewerb-2025-43-Runde-3- Junioraufgabe-2 Sophie-RS

Gebe den Text ein, welcher getrennt werden soll: Ich angle Karpfen.

---Ergebnis-----------------------------
Ich an|gle Kar|pfen.
----------------------------------------
```

Beispiel07: Was sind acht Hobbits? Ein Hobbyte!

```
-Jugendwettbewerb-2025-43-Runde-3- Junioraufgabe-2 Sophie-RS

Gebe den Text ein, welcher getrennt werden soll: Was sind acht Hobbits? Ein Hobbyte!

---Ergebnis-----------------------------
Was sind acht Hob|bits? Ein Hobby|te!
----------------------------------------
```

Beispiel08: Freude, schöner Götterfunken, Tochter aus Elisium, Wir betreten feuertrunken, Himmlische, dein Heiligthum. Deine Zauber binden wieder, Was die Mode streng getheilt, Alle Menschen werden Brüder, Wo dein sanfter Flügel weilt.

```
-Jugendwettbewerb-2025-43-Runde-3- Junioraufgabe-2 Sophie-RS

Gebe den Text ein, welcher getrennt werden soll: Freude, schöner Götterfunken, Tochter aus Elisium, Wir betreten feuertrunk
en, Himmlische, dein Heiligthum. Deine Zauber binden wieder, Was die Mode streng getheilt, Alle Menschen werden Brüder, Wo
dein sanfter Flügel weilt.

---Ergebnis-----------------------------
Freu|de, schö|ner Göt|ter|fun|ken, Toch|ter aus E|li|si|um, Wir be|tre|ten feu|er|trun|ken, Himm|li|sche, dein Hei|lig|thum
. Dei|ne Zau|ber bin|den wie|der, Was die Mo|de streng ge|theilt, Al|le Men|schen wer|den Brü|der, Wo dein sanf|ter Flü|gel
 weilt.
----------------------------------------
```

Eigene Beispiele

Beispiel01: Ich habe mich heute impfen lassen und habe dabei Tropfen auf dem Dach gesehen, welche nicht mehr dort waren, als ich fertig war.

```
-Jugendwettbewerb-2025-43-Runde-3- Junioraufgabe-2 Sophie-RS

Gebe den Text ein, welcher getrennt werden soll: Ich habe mich heute impfen lassen und habe dabei Tropfen auf dem Dach gese
hen, welche nicht mehr dort waren, als ich fertig war.

---Ergebnis------------------------------
Ich ha|be mich heu|te im|pfen las|sen und ha|be da|bei Tro|pfen auf dem Dach ge|se|hen, wel|che nicht mehr dort wa|ren, als
 ich fer|tig war.
----------------------------------------
```

Beispiel02: Eine Schranke, die an einer Straße gebaut ist, könnte man auch Autoschranke nennen.

```
-Jugendwettbewerb-2025-43-Runde-3- Junioraufgabe-2 Sophie-RS

Gebe den Text ein, welcher getrennt werden soll: Eine Schranke, die an einer Straße gebaut ist, könnte man auch Autoschrank
e nennen.

---Ergebnis------------------------------
Ei|ne Schran|ke, die an ei|ner Stra|ße ge|baut ist, könn|te man auch Au|to|schran|ke nen|nen.
----------------------------------------
```

Beispiel03: Ich sammelte heute alle Pfandflaschen ein und brachte diesen Gemeinschaftspfand zum Supermarkt.

```
-Jugendwettbewerb-2025-43-Runde-3- Junioraufgabe-2 Sophie-RS

Gebe den Text ein, welcher getrennt werden soll: Ich sammelte heute alle Pfandflaschen ein und brachte diesen Gemeinschafts
pfand zum Supermarkt.

---Ergebnis------------------------------
Ich sam|mel|te heu|te al|le Pfand|fla|schen ein und brach|te die|sen Ge|mein|schafts|pfand zum Su|per|markt.
----------------------------------------
```

Quellcode

Vor definierte Listen

```javascript
// Vokale, Selbstlaute die in jeder Silbe sind.
const vokale = ["a", "e", "i", "o", "u", "ö", "ä", "ü"];
// Diphthongs, ein Doppellaut bestehend aus zwei Vokalen.
const diphthongs = ["au", "ei", "ai", "eu", "äu", "oi", "ui", "ee", "aa", "ie"];
// Digraphen, zwei Buchstaben die ein Laut bilden (Mit einem Trigraphen ('sch')).
const digraphen = ["ck", "ch", "ph", "th", "qu", "sch"];
// Kombination aus den digraphen und diphthongs.
const paare = [...diphthongs, ...digraphen];
// Anlaute, Laute mit welcher eine Silbe beginnen muss.
const anlaute = [
  ["sch"],
  [
    "bl", "pf", "br", "dr", "fl", "fr", "gl", "gr",
    "gn", "kl", "kr", "pl", "pr", "tr", "st", "sp",
    "ch", "ph", "th", "qu",
  ],
  [
    "a", "e", "i", "o", "u", "ä", "ö", "ü", "b",
    "d", "f", "g", "h", "j", "k", "l", "m", "n",
    "p", "r", "s", "t", "v", "w", "z", "ß"
  ],
];
```

Helfer Funktion

```
/* Hilf Funktionen */

function besitzenSilbenVokale(position: number): boolean {
  return (
    wortArray
      .slice(positionSilbe.at(-1) + 1, position + 1)
      .some((b) => vokale.includes(b.toLowerCase())) &&
    wortArray.slice(position + 1).some((b) => vokale.includes(b.toLowerCase()))
  );
}

/**
 * Prüft, ob an einer bestimmten Position (mit optionalem Zusatz) im Wort ein Vokal steht.
 *
 * @param position Die aktuelle Position des Buchstabens, an dem sich das Programm gerade befindet.
 * @param additionToPos Ein optionaler Zusatz zur Position, um eine Verschiebung vorzunehmen.
 * @returns Gibt zurück, ob das Array 'wortArray' an der berechneten Position einen Vokal enthält.
 */
function istVokal(position: number, additionToPos: number = 0): boolean {
  if (!wortArray[position + additionToPos]) return false;

  return vokale.includes(wortArray[position + additionToPos].toLowerCase());
}

/**
 * Prüft, ob sich an einer bestimmten Position im Wort ein Buchstabenpaar befindet oder darauf folgt.
 *
 * @param pos Die aktuelle Position des Buchstabens, an dem sich das Programm gerade befindet.
 * @returns Gibt zurück, ob im Array 'wortArray' an der angegebenen Position ein Buchstabenpaar vorhanden ist oder folgt.
 */
function istPaar(pos: number): boolean {
  if (pos > wortArray.length - 2) return false;

  if (paare.some((value) => value.substring(0, 2) === wortArray.slice(pos, pos + 2).join(""))) {
    if (wortArray.slice(pos, pos + 2000).join("") !== "sc") return true;

    if (wortArray.slice(pos, pos + 3).join("") === "sch") return true;
  } else return false;
}

/**
 * Prüft, ob sich zwischen einer bestimmten Position und dem nächsten Vokal ein Anlaut befindet.
 *
 * @param position Die aktuelle Position des Buchstabens, an dem sich das Programm gerade befindet.
 * @returns Gibt zurück, ob sich im Array 'wortArray' zwischen der angegebenen Position und dem nächsten Vokal ein Anlaut befindet.
 */
function startetMitAnlaut(position: number): boolean {
  if (istVokal(position + 1)) return true;

  let nextVokal = position + 1;

  while (!istVokal(nextVokal) && position < wortArray.length - 1) nextVokal++;

  const abstandZuVokal = nextVokal - 1 - position;

  if (abstandZuVokal <= 3 && abstandZuVokal > 0) {
    return anlaute.flat(2).some((v) => v === wortArray.slice(position + 1, nextVokal).join(""));
  } else if (abstandZuVokal === 4) {
    return (
      wortArray.slice(position + 1, position + 4).join("") === anlaute[0][0] &&
      anlaute[2].some((v) => v === wortArray[position + 4])
    );
  }
  return false;
}
}
```

Regel-loop

```
// Trenne das Wort
for (let i = 0; i < wortArray.length; i++) {
  // 7. Eine Silbe muss einen Vokal enthalten.
  if (!besitzenSilbenVokale(i)) {
    // Silbe enthält kein Vokal -> Nicht trennen
    continue;
  }
  // 6. Führt die Trennung dazu, dass ein Buchstabenpaar
  // (z. B. „sch“, „ch“, „au“ …) getrennt würde,
  // darf hier nicht getrennt werden.
  else if (istPaar(i)) {
    if (wortArray[i] === "s") i++;
    // Ein Paar würde getrennt werden -> Nicht trennen
    continue;
  }
  // 5. Beginnt die neu entstehende Silbe nicht mit
  // einem Anlaut, darf nicht getrennt werden.
  else if (!startetMitAnlaut(i)) {
    // Die neu entstehende Silbe beginnt nicht mit
    // einem Anlaut -> Nicht trennen
    continue;
  }
  // 4. Folgen auf einen Vokal nicht zwei Konsonanten,
  // wird nach dem Vokal getrennt.
  else if (istVokal(i) && (istVokal(i, 1) || istVokal(i, 2) || i === wortArray.length - 2)) {
    positionSilbe.push(i);
    continue;
  }
  // 3. Bei drei oder mehr aufeinanderfolgenden Konsonanten
  //  wird nach dem ersten Konsonanten getrennt.
  else if (i <= wortArray.length - 3 && !istVokal(i) && !istVokal(i, 1) && !istVokal(i, 2)) {
    positionSilbe.push(i);
    continue;
  }
  // 2. Der erste und der letzte Buchstabe dürfen nicht getrennt werden.
  else if (i === 0 || i >= wortArray.length - 2) {
    // Der erste oder letzte Buchstabe würde abgetrennt werden -> Nicht trennen
    continue;
  }
  // 1. Trenne das Wort.
  else {
    positionSilbe.push(i);
    continue;
  }
}
```