

Zeichenkettenverarbeitung mit Snap!¹

Ein- und Ausgabe von Texten

Vielleicht kennen Sie bereits die Blöcke `ask what's your name? and wait` und `answer`. Mit diesen Blöcken können wir eine Eingabe vom Anwender erfragen. Häufig ist diese Eingabe ein Text aus Buchstaben, Ziffern und anderen Zeichen. Ein Text besteht also aus vielen einzelnen Zeichen. Deshalb wird ein Text beim Programmieren Zeichenkette genannt. Auch eine von uns erstellte Variable kann eine Zeichenkette aufnehmen. Im Folgenden wollen wir uns anschauen, wie wir eine solche Zeichenkette verarbeiten und verändern können.

Blöcke zur Verarbeitung von Zeichenketten

Um eine Zeichenkette zu verarbeiten, sind folgende Blöcke aus dem Bereich **Operators** hilfreich.

Block	Bedeutung
<code>join</code> <code>hello</code> <code>world</code>	Verbindet die beiden Zeichenketten „hello“ und „world“ zu einer: „hello world“. Für „hello“ und „world“ können auch Variablen eingesetzt werden, die Zeichenketten oder einzelne Zeichen enthalten.
<code>letter</code> <code>1</code> <code>of</code> <code>world</code>	Liefert den ersten Buchstaben der Zeichenkette, im Beispiel „w“. Hier können auch andere Zahlen angegeben werden.
<code>length</code> <code>of text</code> <code>world</code>	Liefert die Länge der angegebenen Zeichenkette, im Beispiel 5
<code>=</code>	Vergleicht zwei Zeichenketten miteinander. Groß- und Kleinschreibung wird dabei nicht unterschieden.

Tabelle 1: Blöcke zur Verarbeitung von Zeichenketten

In die Blöcke können auch Variablen eingesetzt werden, die eine Zeichenkette enthalten. Abbildung 1 zeigt ein Beispielprogramm. Als Eingabe wurde der Text *Auf der grünen Wiese steht eine Kuh und lächelt* gewählt.

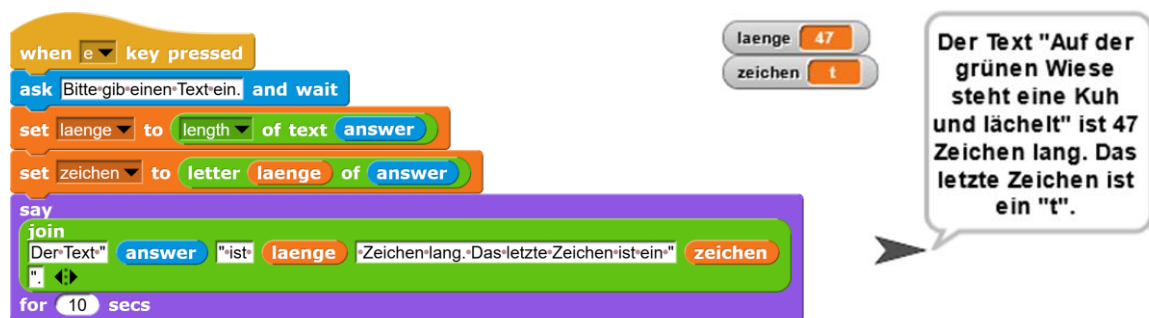


Abbildung 1: Verwendung der Blöcke zur Verarbeitung von Zeichenketten. Beispielprogramm und Ausgabe

Aufgabe 1:

- Ändern Sie das Programm aus Abbildung 1 so, dass das erste, das mittlere und das vorletzte Zeichen angezeigt werden.

¹ Snap! wird von der University of California, Berkeley zur Verfügung gestellt: <https://snap.berkeley.edu> Für die Beispiele wurde Snap! in der Version 6.3.7 verwendet.

- b) Legen Sie ein Codewort fest, das der Anwender eingeben muss, bevor das Programm weiter ausgeführt wird.
- c) Erlauben Sie dem Anwender bei Drücken der Taste 'c' das Codewort zu ändern. Dazu muss er erst das alte Codewort eingeben und darf dann ein neues wählen. Das neue Codewort soll mindestens 8 Zeichen lang sein.

Aufgabe 2: Bei einer Kindersuchmaschine sollen bestimmte Suchwörter nicht erlaubt sein. Entwickeln Sie ein Programm, das die Eingabe des Anwenders nur dann identisch wieder ausgibt, wenn es sich nicht um eines der verbotenen Wörter handelt. Welche Wörter nicht erlaubt sind, dürfen Sie selbst festlegen.

Aufgabe 3: Überlegen Sie sich fünf Fragen für ein kleines Quiz. Der Anwender kann das Quiz z. B. mit der Taste 'q' starten. Ihm werden dann nacheinander die fünf Fragen gestellt, zu denen er in einem Eingabefeld die Antwort eingeben muss. Am Ende erscheint im Programmfenster die Anzahl der richtigen Antworten.

Mögliche Erweiterungen:

- Bei einer falschen Antwort erhält man einen zweiten Versuch.
- Es wird zu jeder Frage angezeigt, ob die Antwort richtig oder falsch war.
- Es kann zwischen zwei Schwierigkeitsstufen gewählt werden.

Zeichenweise Verarbeitung einer Zeichenkette

Für viele algorithmische Probleme, bei denen Zeichenketten überprüft oder verändert werden, ist es notwendig, eine Zeichenkette Zeichen für Zeichen zu betrachten. Das grundsätzliche Vorgehen dazu schauen wir uns an einem Beispiel an. Dieses Grundgerüst kann dann auf andere Probleme übertragen und entsprechend angepasst werden.

Aufgabe 4:

- a) Öffnen Sie das Skript aus Abbildung 2 in Snap! und testen Sie es für die Eingaben Rätsel und ätschibätsch. Betrachten Sie auch einmal die Ausgabe des Programms, die sich ergibt, wenn Sie den say-Block aus Zeile 9 in die for-Schleife unter die Verzweigung verschieben.
- b) Erstellen Sie für das Skript in Abbildung 2 und die Eingabe Bär eine Tracetabelle (s. Tabelle 2). Formulieren Sie in Ihren eigenen Worten, wie das Skript die Eingabe verändert.

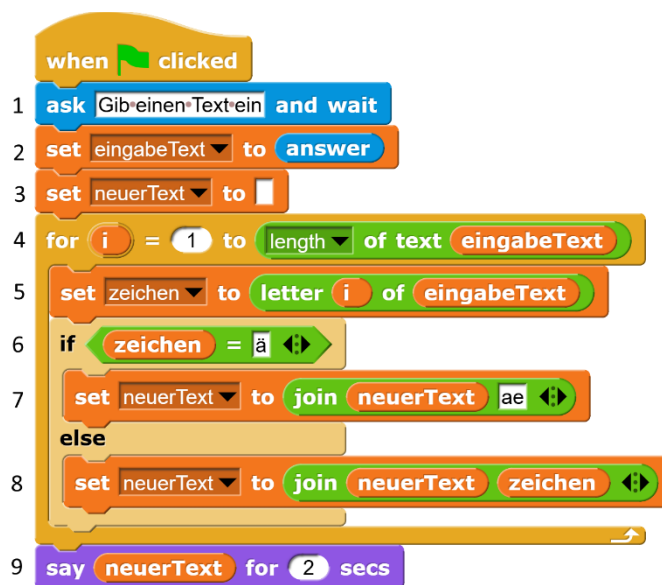


Abbildung 2: Beispiel für die zeichenweise Verarbeitung einer Zeichenkette

Zeile	eingabeText	neuerText	i	zeichen	zeichen = ä
2	Bär				
3		""			
...		

Tabelle 2: Tracetabelle für das Programm in Abbildung 2

Wenn wir die Eingabe mit der Ausgabe vergleichen, sehen wir, dass in der eingegebenen Zeichenkette alle 'ä' durch "ae" ersetzt wurden. Da die Umlaute 'ä', 'ö' und 'ü' nicht in allen Sprachen bekannt sind, ist das manchmal notwendig. Wie wir an der Tracetabelle sehen, wurden die Zeichen aber nicht einfach ersetzt, sondern die neue Zeichenkette wurde Zeichen für Zeichen aufgebaut. Dort wo vorher ein 'ä' stand, wurde "ae" angefügt und ansonsten wurde das jeweilige Zeichen aus der alten Zeichenkette übernommen. Dazu wird in Zeile 3 vor Beginn der Schleife eine neue Variable `neuerText` erzeugt, die zunächst leer ist. Die Variable `i` in der `for`-Schleife startet bei dem Wert 1 und wird dann in jedem Schleifendurchlauf um eins erhöht, um die einzelnen Positionen der Zeichen durchzugehen. Der maximale Wert von `i` entspricht der Länge des zu durchlaufenden Zeichenkette. In der Schleife wird für jedes Zeichen überprüft, ob es sich um ein 'ä' handelt. Wenn dies der Fall ist, wird "ae" an die Variable `neuerText` gehängt, ansonsten das Zeichen selbst. Bei der Zuweisung an die Variable `neuerText` ist wichtig, dass der bisherige Inhalt von `neuerText` und das neue Zeichen zunächst mit dem `join`-Block verbunden werden, da der bisherige Inhalt sonst jedes Mal verloren ginge.

Es ist sehr ratsam, den ursprünglichen Eingabetext innerhalb der Schleife nicht zu verändern. Denn wenn wir ein 'ä' durch ein ae ersetzen oder andersherum, verändert sich die Positionszahl aller nachfolgenden Zeichen und die Länge der Zeichenkette. Es könnten dadurch Zeichen vergessen oder doppelt betrachtet werden. Deshalb ist man auf der sicheren Seite, wenn man den neuen Text in einer neuen Variablen aufbaut.

Aufgabe 5:

- Erweitern Sie das Beispiel aus Abbildung 2 so, dass auch alle 'ö' und 'ü' durch "oe" bzw. "ue" ersetzt werden.
- Anhand des Beispiels aus Abbildung 2 wurde erläutert, wie eine Zeichenkette, die in einer Variablen gespeichert ist, verändert werden kann, indem sie Zeichen für Zeichen durchlaufen und das Ergebnis in einer neuen Variablen aufgebaut wird. Dazu können die Fragmente aus Abbildung 3 verwendet werden. Identifizieren Sie diese Fragmente in dem Skript aus Abbildung 2. Welche Variablen entsprechen den Variablen `eingabe` und `ausgabe`, welcher Teil des Skriptes aus Abbildung 2 ersetzt den `TODO` Block?
- Statt der `for`-Schleife könnten auch eine `repeat`-Schleife oder eine `repeat until`-Schleife verwendet werden. Wie müsste die Bedingung für den Schleifenabbruch dann jeweils aussehen? Werden zusätzliche Variablen benötigt?



Abbildung 3: Codefragmente zur Verarbeitung von Zeichenketten

- d) Ändern Sie das Skript in Abbildung 2 so, dass eine *repeat*-Schleife oder eine *repeat until*-Schleife verwendet werden.

Aufgabe 6:

- a) Wählen Sie ein Sprite aus. Das Sprite soll nach einem Wort fragen und dieses dann Zeichen für Zeichen buchstabieren.
b) Das Sprite soll den Benutzer wieder nach einem Wort fragen und diesmal rückwärts ausgeben.

Aufgabe 7:

- a) Es gibt die Empfehlung, einstellige Zahlen in Texten auszuschreiben. Erstellen Sie ein Programm, das die Zahlen 0 bis 9 in einem Eingabetext durch die entsprechenden Zahlwörter ersetzt. Sie können hier zunächst davon ausgehen, dass der Eingabetext keine mehrstelligen Zahlen enthält.
b) Eine komplexere Variante, die auch der Empfehlung nachkommt, die Zahlen 10, 11 und 12 auszuschreiben und Ziffern nicht ersetzt, wenn sie Teil einer größeren Zahl sind, ist deutlich schwieriger, aber nicht unmöglich. Probieren Sie aus, wie weit Sie mit der Lösung kommen.

Dafür kann der Block  hilfreich sein.

Aufgabe 8: Erweitern Sie Ihre Lösung zu Aufgabe 1 so, dass der Anwender bei der Wahl seines Codeworts mindestens eines der Sonderzeichen #, *, +, ! oder ? verwenden muss. Ansonsten wird er zur Wahl eines anderen Codeworts aufgefordert.

Aufgabe 9: Eine Möglichkeit, einen Text unleserlich zu machen, ist, die Reihenfolge der Buchstaben zu vertauschen.

- a) Erstellen Sie ein Programm, das die Buchstaben so vertauscht, dass erst alle Zeichen an einer ungeraden Position (1, 3, 5, ...) und dann alle Zeichen an einer geraden Position (2, 4, 6, ...) ausgegeben werden. Aus dem Wort *Apfelmus* würde so z. B. *Aflupems*.
b) Erweitern Sie Ihr Programm aus a) um die Option, die Vertauschung wieder rückgängig zu machen.
c) Überlegen Sie sich weitere Regeln für die Vertauschung der Buchstaben und erstellen Sie ein entsprechendes Programm.

Aufgabe 10: Beim Knacken von Geheimschriften kann es hilfreich sein, die Anzahl eines bestimmten Zeichens in einem Text zu bestimmen. Dabei müssen Klein- und Großbuchstaben nicht unterschieden werden.

- a) Erstellen Sie ein Programm, das ausgibt, wie oft der Buchstabe A in einem Text, den der Anwender eingibt, vorkommt.
b) Erstellen Sie ein Programm, bei dem der Anwender zunächst einen Buchstaben auswählen kann, der gezählt werden soll. Anschließend gibt er einen Text ein, in dem der entsprechende Buchstabe gezählt werden soll.
c) Erstellen Sie ein Programm, das für den Text, den der Anwender eingibt, ermittelt, welcher Vokal am häufigsten vorkommt.

Ausblick: Ein Programm, das allgemein den häufigsten Buchstaben in einem Text ermittelt, erstellen Sie in Aufgabe 15. Dazu ist es hilfreich sich im Folgenden zunächst etwas genauer mit der internen Codierung von Zeichen zu beschäftigen.

Verarbeitung einzelner Zeichen mithilfe des Unicode

Da der Computer intern nur mit Binärzahlen arbeiten kann, müssen alle Zeichen binär codiert werden. Dazu wird jedem Zeichen eine Zahl zugeordnet. Snap! verwendet zur Codierung von Zeichen den weit verbreiteten Unicode. Tabelle 3 zeigt einen Ausschnitt der Zuordnung von Zeichen zu Zahlen gemäß dem Unicode. Zur besseren Lesbarkeit sind die Zahlen hier im Dezimalsystem dargestellt. Es fällt auf, dass sowohl die Ziffern von 0 bis 9 als auch die Klein- und Großbuchstaben jeweils fortlaufend durchnummeriert sind. Wir können uns den Unicode daher zu Nutze machen, um die Zeichen des Alphabets systematisch zu durchlaufen, Groß- und Kleinbuchstaben zu unterscheiden oder mit Zeichen "zu rechnen". Das schauen wir uns im Folgenden genauer an.

Zeichen	Unicode	Zeichen	Unicode	Zeichen	Unicode
!	33	2	50	D	68
#	35
*	42	9	57	Z	90
+	43	A	65	a	97
0	48	B	66
1	49	C	67	z	122

Tabelle 3: Ausschnitt des Unicode

Um ein Zeichen in den zugehörigen Unicode umzuwandeln und umgekehrt stehen in Snap! zwei Blöcke zur Verfügung:



Block	Bedeutung
	Liefert den Unicode des Buchstabens. Achtung: Klein- und Großbuchstaben müssen hier unterschieden werden.
	Liefert den Buchstaben, der zu dem angegebenen Unicode gehört.

Tabelle 4: Blöcke zur Umwandlung von Zeichen in Unicode und umgekehrt

Aufgabe 11:

- Stellen Sie anhand der Tabelle 3, die einen Ausschnitt des Unicode zeigt, eine Vermutung auf, welche Zahl jeweils die folgenden Zeichen codiert: 3, 6, E, M, e, m
Überprüfen Sie Ihre Vermutung anschließend mithilfe der Blöcke aus Tabelle 4.
- Verwenden Sie die Blöcke aus Tabelle 4, um auch für die folgenden Zeichen den Unicode zu ermitteln: @ = ! ; (

Aufgabe 12:

- Erstellen Sie ein Programm, das den Anwender nach einer Zahl des Unicode fragt und das passende Zeichen dazu ausgibt.
- Verbessern Sie Ihr Programm aus Aufgabenteil a), indem Sie überprüfen, ob es sich bei den eingegebenen Zeichen nur um Ziffern handelt. Prüfen Sie zusätzlich, ob es sich bei der Eingabe um eine Zahl größer als 33 handelt, da alle kleineren Zahlen Steuerzeichen codieren, die nicht angezeigt werden können. Geben Sie ansonsten entsprechende Fehlermeldungen aus.

Aufgabe 13:

- Erstellen Sie ein Programm, das für den Text, den der Anwender eingibt, die Codierung mit dem Unicode ausgibt. Setzen Sie zwischen die Zahlen des Unicodes jeweils ein Leerzeichen.
Beispiel: Für die Eingabe „Hallo“ ist die Ausgabe 72 97 108 108 111
- Überlegen Sie, ob es sich hierbei um eine Verschlüsselung handelt. Begründen Sie Ihre Antwort.
- Erweitern Sie Ihr Programm so, dass auch eine Decodierung von der Zahlendarstellung im Unicode in Buchstaben möglich ist. Gehen Sie davon aus, dass nur gültige Unicode-Werte getrennt durch Leerzeichen eingegeben werden, z. B. 73 110 102 111 114 109 97 116 105 107

Aufgabe 14: Erweitern Sie Ihr Programm aus Aufgabe 5 so, dass beim Ersetzen der Umlaute zwischen Klein- und Großbuchstaben unterschieden wird. Das heißt ein kleines 'ä' soll durch "ae" und ein großes 'Ä' durch "Ae" ersetzt werden.

Aufgabe 15:

Erstellen Sie ein Programm, das für den Text, den der Anwender eingibt, ermittelt, welcher Buchstabe am häufigsten vorkommt. Gibt es mehrere häufigste Buchstaben, reicht die Ausgabe eines häufigsten Buchstabens.

Hinweis 1: Wandeln Sie den eingegebenen Text zunächst in Großbuchstaben um. Verwenden Sie dazu den Block „upper case of text ...“ (s. Abbildung 4).

Hinweis 2: Verwenden Sie den Unicode, um in einer Schleife nacheinander die Häufigkeit aller Großbuchstaben zu bestimmen.

length ▼ of text world

upper case ▼ of text world

Abbildung 4: Bei dem Block "length ..." können Sie alternativ die Funktion "upper case ..." auswählen.

Aufgabe 16:

Bei der Caesar-Verschlüsselung werden die Zeichen des Klartextes um eine bestimmte Anzahl an Stellen im Alphabet verschoben, um den Geheimtext zu erhalten. Um die Ver- und Entschlüsselung per Hand durchzuführen kann z. B. eine Caesar-Scheibe verwendet werden. Im Folgenden soll die Verschiebung algorithmisch umgesetzt werden.

Verwenden Sie für die folgenden Aufgaben zur Vereinfachung nur Großbuchstaben bei der Eingabe.

- Erstellen Sie als Vorübung ein Programm, das den zehnten Buchstaben im Alphabet nach dem eingegebenen Buchstaben ausgibt. Wenn der Buchstabe 'Z' erreicht wurde, soll wieder bei 'A' begonnen werden.
- Erstellen Sie ein Programm, das als Eingabe den Klartext und einen Schlüssel, der die Verschiebung angibt, erhält und dazu den Geheimtext ausgibt. Überlegen Sie sich geeignete Eingaben zum Testen. Überlegen Sie auch, wie sie mit Satzzeichen und Leerzeichen verfahren.
- Erweitern Sie Ihr Programm, um die Möglichkeit der Entschlüsselung. Als Eingaben benötigen Sie dazu den Geheimtext und den Schlüssel.
- Verwenden Sie Ihr Programm aus Aufgabe 15, um einen Geheimtext, der nach dem Caesar-Verfahren verschlüsselt wurde, zu knacken. Das heißt, das Programm soll zu einem Geheimtext ohne Kenntnis des Schlüssels einen wahrscheinlichen Vorschlag für den Klartext machen.

Projekt:

Einzelarbeit: Informieren Sie sich über verschiedene Verschlüsselungsverfahren. Entscheiden Sie sich für ein Verfahren, das Sie algorithmisch umsetzen. Beispiele wären das Vigenère-Verfahren oder die Skytale. Sie können sich auch selbst ein Verfahren ausdenken!

Gruppenarbeit: Informieren Sie sich über verschiedene Verschlüsselungsverfahren oder entwerfen Sie selbst verschiedene Verfahren. Setzen Sie die Verschlüsselungsverfahren algorithmisch um. Erstellen Sie in der Gruppe ein umfangreicheres Programm, das mehrere Verschlüsselungsverfahren anbietet. Teilen Sie die Implementierung der ausgewählten Verfahren unter sich auf.

Ausblick: Weitere Operationen für Zeichenketten

Für Zeichenketten stehen eine Reihe weiterer Operationen zur Verfügung, die bei der Verarbeitung von Zeichenketten hilfreich sein können. Exemplarisch betrachten wir hier die Operationen *substring* und *contains*. Tabelle 4 gibt einen Überblick. Die Beispiele verwenden eine Variable `text` vom Typ *String*, in der die Zeichenkette "Hallo Siri" gespeichert ist. Um die entsprechenden Blöcke verwenden zu können, muss in Snap! die Bibliothek "Strings, Multi-line input" importiert werden.

Block / Beispiele	Bedeutung
<p>The screenshot shows two main sections of Snap! blocks. The top section, titled 'substring of', shows three examples: 1) A block with empty fields for 'from position' and 'to position'. 2) A block with 'HalloSiri' in the 'substring of' field, '1' in 'from position', '5' in 'to position', and 'inclusive' checked. A speech bubble shows 'Hallo'. 3) A block with 'HalloSiri' in the 'substring of' field, '7' in 'from position', '8' in 'to position', and 'inclusive' checked. A speech bubble shows 'Si'. The bottom section, titled 'text contains', shows four examples: 1) A block with empty fields for 'text' and 'contains'. 2) A block with 'HalloSiri' in 'text' and 'Siri' in 'contains', with a 'true' speech bubble. 3) A block with 'HalloSiri' in 'text' and 'Alexa' in 'contains', with a 'false' speech bubble. 4) A block with 'HalloSiri' in 'text' and 'A' in 'contains', with a 'true' speech bubble.</p>	<p>Auslesen einer Teilzeichenkette in einem bestimmten Bereich.</p> <p>Prüfen einer Zeichenkette auf eine Teilzeichenkette.</p> <p>Der Rückgabewert ist <code>true</code>, wenn die Zeichenkette, die als Parameter übergebene Zeichenfolge enthält, sonst <code>false</code>.</p> <p>Hinweis: Klein- und Großbuchstaben werden dabei nicht unterschieden.</p>

Tabelle 5: Die Blöcke *substring* und *contains*

Aufgabe 17: Eine IBAN ist nach einem festen Schema aufgebaut:

2-stellige 10-stellige
Prüfziffer Kontonummer

DE22100100500123456789

2-stellige 8-stellige
Länderkennung Bankleitzahl

Abbildung 5: Aufbau einer IBAN

Erstellen Sie ein Programm, das eine IBAN, die der Anwender eingibt, in die einzelnen Bestandteile zerlegt. Eine Figur kann die einzelnen Bestandteile nacheinander auf der Bühne anzeigen (s. Abbildung 6)



Abbildung 6: mögliche Ausgabe

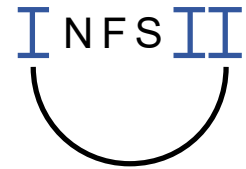
Aufgabe 18: Manche Spamfilter ordnen Mails als Spam ein, wenn sie bestimmte auffällige Wörter enthalten. Überlegen Sie sich vier Wörter, die auf eine Spam-Mail hinweisen könnten. Erstellen Sie anschließend ein Programm, das einen Text darauf prüft, ob eines dieser auffälligen Wörter enthalten ist und eine entsprechende Rückmeldung gibt, ob es sich um eine Spam-Mail handeln könnte.

Aufgabe 19: Plattformen, auf denen Bilder hochgeladen werden können, schränken manchmal die erlaubten Dateiformate ein. Erlaubt sein könnten z. B. die Formate "jpg" und "png".

Erstellen Sie ein Programm, das prüft, ob ein Dateiname, der eingegeben wird, mit einem gültigen Dateiformat endet und dem Anwender eine entsprechende Rückmeldung gibt.

Aufgabe 20:

- Betrachten Sie noch einmal die Aufgaben 2 und 3 am Beginn des Leitfadens. Diskutieren Sie, ob die Verwendung des Blocks `text ... contains ...` statt des Blocks `... = ...` die Implementierung eines entsprechenden Programms erleichtert.
- In Aufgabe 7 sollte ein vom Anwender gewähltes Codewort darauf überprüft werden, ob es eines der Sonderzeichen #, *, +, ! oder ? enthält. Erläutern Sie, wie sich die Implementierung unter Verwendung des Blocks `text ... contains ...` vereinfacht.



Lizenz

Dieses Werk ist lizenziert unter einer [Creative Commons Namensnennung - Nicht-kommerziell - Weitergabe unter gleichen Bedingungen 4.0 International Lizenz](#). Von der Lizenz ausgenommen ist das InfSII-Logo.

Für die korrekte Ausführbarkeit der Quelltexte in diesem wird keine Garantie übernommen. Auch für Folgeschäden, die sich aus der Anwendung der Quelltexte oder durch eventuelle fehlerhafte Angaben ergeben, wird keine Haftung oder juristische Verantwortung übernommen.