## 1.10 Zeichenketten: Texte Zeichen für Zeichen

**Leitidee**

Namen, Passwörter, Nachrichten – sehr viele Daten in Programmen sind Texte. Bisher hast du Texte nur als Ganzes benutzt: eingelesen, ausgegeben, mit `+` zusammengesetzt. Oft muss ein Programm aber **in einen Text hineinschauen**: Wie lang ist das Passwort? Enthält es eine Ziffer? Mit welchem Buchstaben beginnt der Name? Wie viele Vokale stecken in einem Satz? Dafür lernst du in diesem Abschnitt, wie Python einen Text sieht: als eine Reihe einzelner Zeichen, jedes an einer festen Position. Mit diesem Wissen kannst du später sogar Texte verschlüsseln.

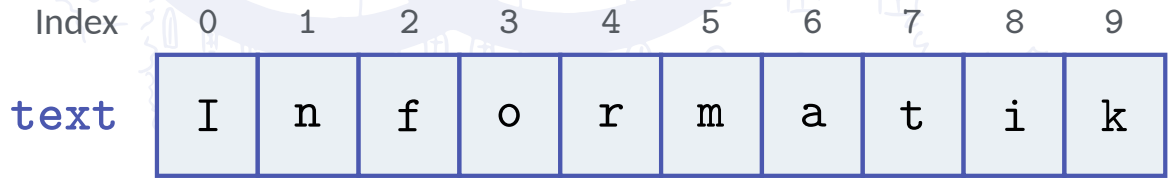
**Lernziele**

Nach diesem Abschnitt kannst du

- eine Zeichenkette als Folge (Sequenz) von Zeichen beschreiben und den Index eines Zeichens angeben,
- mit `text[index]` auf einzelne Zeichen zugreifen und mit `len(text)` die Länge bestimmen,
- Zeichenketten verketten und Ausgaben mit f-Strings bequem zusammensetzen,
- mit einer Schleife alle Zeichen einer Zeichenkette durchlaufen,
- `in`, `.upper()`, `.lower()`, `.isalpha()` und `.isdigit()` anwenden,
- einen `IndexError` erklären und vermeiden.

### 1.10.1 Eine Zeichenkette ist eine Folge von Zeichen

Für Python besteht ein Text aus einzelnen **Zeichen**, die der Reihe nach hintereinander stehen – wie Perlen auf einer Schnur. Ein Zeichen ist ein Buchstabe, eine Ziffer, ein Satzzeichen oder auch ein Leerzeichen. Jedes Zeichen hat eine feste Position. Diese Positionsnummer heißt **Index**. Gezählt wird ab **0**, nicht ab 1:




Die Zeichenkette `"Informatik"` in der Variablen `text`: Jedes Zeichen hat eine Position (Index), gezählt ab 0

Das erste Zeichen `I` hat also den Index 0, das zweite Zeichen `n` den Index 1 und das letzte Zeichen `k` den Index 9. Die Zeichenkette hat 10 Zeichen, aber der größte Index ist 9. Warum ab 0? Man kann sich den Index als Abstand vom Anfang vorstellen: Das erste Zeichen ist 0 Schritte vom Anfang entfernt.

**Definition: Zeichenkette**

Eine **Zeichenkette** (englisch *string*, Datentyp `str`) ist eine Folge (**Sequenz**) von Zeichen, z. B. `"Informatik"`. Die Zeichen stehen in einer festen Reihenfolge.

**Definition: Index**

Der **Index** gibt die Position eines Zeichens in der Zeichenkette an. Das erste Zeichen hat den Index 0, das zweite den Index 1 und so weiter.

**Merke**

- Eine **Zeichenkette** (`str`) ist eine **Sequenz von Zeichen**, z. B. `"Informatik"`.
- Jedes Zeichen hat eine **Position (Index)**, beginnend bei `0`.

### 1.10.2 Auf einzelne Zeichen zugreifen

Um ein einzelnes Zeichen zu bekommen, schreibst du hinter die Zeichenkette den Index in **eckigen Klammern**:

```
1 text = "Informatik"
2 print(text[0])
3 print(text[2])
4 print(text[9])
```

AUSGABE

```
I
f
k
```

- **Zeile 1:** Die Variable `text` bekommt die Zeichenkette `"Informatik"`.
- **Zeile 2:** `text[0]` ist das Zeichen mit dem Index 0, also das erste Zeichen: `I`.
- **Zeile 3:** `text[2]` ist das **dritte** Zeichen, `f`. Schau in der Abbildung oben nach.
- **Zeile 4:** `text[9]` ist das letzte Zeichen, `k`.

In den eckigen Klammern darf auch eine Variable oder eine Rechnung stehen, zum Beispiel `text[i]` oder `text[i + 1]`. Das Ergebnis ist selbst wieder eine Zeichenkette, die aus genau einem Zeichen besteht.

**Merke**

Zugriff auf ein einzelnes Zeichen: `text[index]`.

### 1.10.3 Die Länge einer Zeichenkette

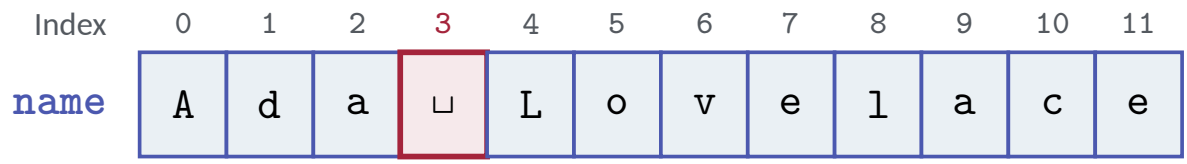
Wie viele Zeichen eine Zeichenkette hat, verrät dir die Funktion `len` (von englisch *length*, „Länge“):

```
1 text = "Informatik"
2 print(len(text))
3 print(len("Ada Lovelace"))
4 print(len(""))
```

AUSGABE

```
10
12
0
```

- **Zeile 2:** `"Informatik"` hat 10 Zeichen.
- **Zeile 3:** `"Ada Lovelace"` hat 12 Zeichen – das Leerzeichen zählt mit (siehe Abbildung). Ada Lovelace gilt übrigens als erste Programmiererin der Welt.
- **Zeile 4:** `""` ist die **leere Zeichenkette**. Sie enthält kein einziges Zeichen und hat die Länge 0.




Auch das Leerzeichen ist ein Zeichen: In `"Ada Lovelace"` steht es am Index 3. Die Zeichenkette hat 12 Zeichen

Weil der Index bei 0 beginnt, hat das **letzte Zeichen** immer den Index `len(text) - 1`. So findest du es, auch wenn du nicht weißt, wie lang der Text ist – zum Beispiel bei einer Eingabe:

```
1 text = "Informatik"
2 letzter_index = len(text) - 1
3 print(letzter_index)
4 print(text[letzter_index])
```

AUSGABE

```
9
k
```

**Merke**

Die **Länge** einer Zeichenkette erhält man mit `len(text)`. Deshalb hat das letzte Zeichen immer den Index `len(text) - 1`.

### Von hinten zählen

Python erlaubt auch **negative Indizes**. Sie zählen von hinten: `text[-1]` ist das letzte Zeichen, `text[-2]` das vorletzte. Bei `"Informatik"` ist `text[-1]` also `k` und `text[-2]` ist `i`. `text[-1]` ist eine kurze Schreibweise für `text[len(text) - 1]`.

### Wenn der Index zu groß ist

Was passiert, wenn du auf ein Zeichen zugreifst, das es gar nicht gibt?

```
text = "Informatik"
print(text[10])
```

FEHLERMELDUNG

```
IndexError: string index out of range
```

**Stolperstein**

Ein **IndexError** bedeutet: „Der Index liegt außerhalb des gültigen Bereichs.“ `"Informatik"` hat 10 Zeichen, gültig sind also nur die Indizes 0 bis 9 (und −1 bis −10). Der Index 10 gehört schon nicht mehr dazu. Das ist ein sehr häufiger Fehler: Man denkt an die Länge 10 und vergisst, dass ab 0 gezählt wird. Prüfe bei einem IndexError deshalb: Wie lang ist der Text, und welcher Index wurde benutzt? Auch bei der leeren Zeichenkette `""` führt schon `text[0]` zu diesem Fehler.

**Merke**

Ein Zugriff auf einen ungültigen Index (z. B. `text[100]` bei kurzem Text) führt zu einer **Fehlermeldung** (`IndexError`).

**Arbeitsauftrag 118: Zeichen bestimmen**

Gegeben ist das folgende Programm. Gib **ohne** Computer an, was jede Zeile ausgibt. Eine Zeile führt zu einem Fehler – welche, und warum? Überprüfe danach am Computer.

```
1 wort = "Python"
2 print(wort[1])
3 print(len(wort))
4 print(wort[len(wort) - 1])
5 print(wort[-2])
6 print(wort[6])
```

Meine Vorhersage:

### 1.10.4 Zeichenketten verketten

Aus dem Abschnitt über Variablen kennst du schon das **Verketten** mit `+`: Zwei Zeichenketten werden zu einer neuen, längeren Zeichenkette aneinandergehängt.

```
1 vorname = "Ada"
2 nachname = "Lovelace"
3 name = vorname + " " + nachname
4 print(name)
5 initialen = vorname[0] + nachname[0]
6 print(initialen)
```

AUSGABE

```
Ada Lovelace
AL
```

- **Zeile 3:** Drei Zeichenketten werden verkettet: der Vorname, ein Leerzeichen und der Nachname. Ohne das `" "` in der Mitte stünde `AdaLovelace` da.
- **Zeile 5:** Verkettung und Index-Zugriff lassen sich kombinieren: Das erste Zeichen des Vornamens und das erste Zeichen des Nachnamens ergeben die Initialen.

**Merke**

Die **Verkettung** (Konkatenation) zweier Zeichenketten erfolgt mit `+`.

**Werkzeug: f-Strings**

Texte mit Zahlen mit `+` zusammenzusetzen ist mühsam: Jede Zahl muss mit `str(...)` umgewandelt werden, und die Leerzeichen vergisst man leicht. Bequemer geht es mit einem **f-String**. Du schreibst ein `f` direkt vor das erste Anführungszeichen. Im Text darfst du dann Variablen oder Rechnungen in **geschweifte Klammern** `{...}` setzen. Python ersetzt sie durch ihren Wert:

```
1 name = "Alex"
2 alter = 15
3 print(f"{name} ist {alter} Jahre alt.")
4 print(name + " ist " + str(alter) + " Jahre alt.")
5 print(f"{name} hat {len(name)} Buchstaben.")
```

AUSGABE

```
Alex ist 15 Jahre alt.
Alex ist 15 Jahre alt.
Alex hat 4 Buchstaben.
```

Zeile 3 und Zeile 4 erzeugen dieselbe Ausgabe, aber der f-String kommt ohne `str(...)` aus. Vergisst du das `f`, erscheinen die geschweiften Klammern unverändert in der Ausgabe: `{name} ist {alter} Jahre alt.`

### 1.10.5 Alle Zeichen durchlaufen

Mit einer `for`-Schleife kannst du nacheinander **jedes Zeichen** einer Zeichenkette bearbeiten. Statt `range(...)` schreibst du hinter `in` einfach die Zeichenkette:

```
for zeichen in "Bit":
    print(zeichen)
```

AUSGABE

```
B
i
t
```

Die Schleifenvariable `zeichen` bekommt im ersten Durchlauf den Wert `"B"`, im zweiten `"i"` und im dritten `"t"`. Danach gibt es keine Zeichen mehr, und die Schleife endet. Der Name `zeichen` ist frei gewählt – du könntest auch `buchstabe` schreiben.

#### Zeichen zählen

Eine typische Aufgabe: Zähle, wie oft ein bestimmtes Zeichen vorkommt. Dazu verbindest du die Schleife mit einer Verzweigung und einer Zählvariablen:

```
1 wort = "Anna"
2 anzahl = 0
3 for zeichen in wort:
4     if zeichen == "n":
5         anzahl = anzahl + 1
6 print("Anzahl n:", anzahl)
```

AUSGABE
```
Anzahl n: 2
```

- **Zeile 2:** Die Zählvariable `anzahl` startet bei 0.
- **Zeile 3:** Die Schleife läuft über alle vier Zeichen von `"Anna"`.
- **Zeilen 4–5:** Nur wenn das aktuelle Zeichen ein `n` ist, wird `anzahl` um 1 erhöht.
- **Zeile 6:** Erst nach der Schleife (nicht eingerückt) wird das Ergebnis ausgegeben.

Das Ablaufprotokoll zeigt, wie `zeichen` nacheinander jeden Wert annimmt und `anzahl` nur bei einem `n` wächst:

| Schritt | Zeile | Anweisung | zeichen | anzahl |
|---|---|---|---|---|
| 1 | 1 | `wort = "Anna"` | – | – |
| 2 | 2 | `anzahl = 0` | – | 0 |
| 3 | 3 | `for zeichen in wort:` | "A" | 0 |
| 4 | 4 | `if zeichen == "n":` | "A" | 0 |
| 5 | 3 | `for zeichen in wort:` | "n" | 0 |
| 6 | 4 | `if zeichen == "n":` | "n" | 0 |
| 7 | 5 | `anzahl = anzahl + 1` | "n" | 1 |
| 8 | 3 | `for zeichen in wort:` | "n" | 1 |
| 9 | 4 | `if zeichen == "n":` | "n" | 1 |
| 10 | 5 | `anzahl = anzahl + 1` | "n" | 2 |
| 11 | 3 | `for zeichen in wort:` | "a" | 2 |
| 12 | 4 | `if zeichen == "n":` | "a" | 2 |
| 13 | 3 | `for zeichen in wort:` | "a" | 2 |
| 14 | 6 | `print("Anzahl n:", anzahl)` | "a" | 2 |

Beachte: Das große `A` am Anfang wird **nicht** gezählt, denn `"A"` und `"a"` sind für Python verschiedene Zeichen.

### Einen neuen Text aufbauen

In einer Schleife kann man auch Zeichen für Zeichen eine **neue** Zeichenkette zusammensetzen. Man beginnt mit der leeren Zeichenkette `""` und hängt in jedem Durchlauf etwas an:

```
1 neu = ""
2 for zeichen in "Bit":
3     neu = neu + zeichen + zeichen
4 print(neu)
```

AUSGABE
```
BBiitt
```

Jedes Zeichen wird doppelt angehängt: Aus `""` wird `"BB"`, dann `"BBii"` und schließlich `"BBiitt"`.

#### Schleife über die Indizes

Manchmal brauchst du nicht nur das Zeichen, sondern auch seine **Position**. Dann lässt du die Schleife über alle Indizes laufen. `range(len(text))` liefert genau die gültigen Indizes 0, 1, ..., `len(text) - 1`:

```
1 text = "Bit"
2 for i in range(len(text)):
3     print(i, text[i])
```

AUSGABE

```
0 B
1 i
2 t
```

- **Zeile 2:** `len(text)` ist 3, also durchläuft `i` die Werte 0, 1 und 2.
- **Zeile 3:** Mit `text[i]` holst du das Zeichen an der Position `i`.

**Merke**

- `for zeichen in text:` durchläuft nacheinander alle **Zeichen** der Zeichenkette.
- `for i in range(len(text)):` durchläuft alle **Indizes**. Das Zeichen dazu ist `text[i]`.

### 1.10.6 Ist ein Zeichen enthalten?

Mit dem Wort `in` prüfst du, ob ein Zeichen – oder ein ganzes Textstück – in einer Zeichenkette vorkommt. Das Ergebnis ist `True` oder `False`:

```
1 print("a" in "Banane")
2 print("x" in "Banane")
3 print("nan" in "Banane")
4 print("B" in "banane")
```

AUSGABE

```
True
False
True
False
```

Die letzte Zeile zeigt wieder: Groß- und Kleinbuchstaben sind verschieden. Besonders praktisch ist `in` mit einer Zeichenkette, die alle „erlaubten“ Zeichen enthält. So prüfst du zum Beispiel, ob ein Buchstabe ein Vokal ist:

```
1 buchstabe = "e"
2 if buchstabe in "aeiou":
3     print(buchstabe, "ist ein Vokal.")
```

AUSGABE

```
e ist ein Vokal.
```

### 1.10.7 Groß- und Kleinschreibung

Für Python sind `"Ja"`, `"ja"` und `"JA"` drei verschiedene Zeichenketten. Oft ist die Schreibweise aber egal. Dann hilft es, alles in Klein- oder Großbuchstaben umzuwandeln:

```
1 text = "Hallo Welt"
2 print(text.upper())
3 print(text.lower())
4 print(text)
```

AUSGABE

```
HALLO WELT
hallo welt
Hallo Welt
```

- **Zeile 2:** `text.upper()` liefert den Text in Großbuchstaben (englisch *upper case*).
- **Zeile 3:** `text.lower()` liefert ihn in Kleinbuchstaben (*lower case*).
- **Zeile 4:** `text` selbst ist **unverändert**. `.upper()` und `.lower()` erzeugen eine neue Zeichenkette und lassen die alte, wie sie ist.

Die Schreibweise mit dem Punkt ist neu: `upper` wird nicht wie `len(text)` mit dem Text in Klammern aufgerufen, sondern **hinter** dem Text, verbunden durch einen Punkt. Solche Funktionen nennt man **Methoden**.

**Definition: Methode**

Eine **Methode** ist eine Funktion, die zu einem Wert gehört. Man ruft sie mit einem Punkt hinter dem Wert auf: `text.upper()`. Die Klammern dürfen auch leer sein, aber sie dürfen nicht fehlen.

Praktisch ist das bei Eingaben: Mit `if antwort.lower() == "ja":` erkennt ein Programm die Antwort, egal ob jemand `ja`, `Ja` oder `JA` eintippt.

**Stolperstein**

Eine Zeichenkette lässt sich nicht verändern. `text.lower()` allein ändert `text` deshalb **nicht**:

```
1 text = "Hallo"
2 text.lower()
3 print(text)
```

AUSGABE

```
Hallo
```

Das Ergebnis von `text.lower()` wurde nirgends gespeichert und ist verloren. Richtig ist `text = text.lower()`: Die Variable `text` bekommt dann die neue, kleingeschriebene Zeichenkette.

### 1.10.8 Zeichen prüfen: Buchstabe oder Ziffer?

Zwei weitere Methoden prüfen, woraus eine Zeichenkette besteht. Sie liefern `True` oder `False`:

- `text.isalpha()` ist `True`, wenn **alle** Zeichen Buchstaben sind (auch Umlaute und ß),
- `text.isdigit()` ist `True`, wenn **alle** Zeichen Ziffern sind.

```
1 print("Hallo".isalpha())
2 print("Hallo Welt".isalpha())
3 print("2026".isdigit())
4 print("-5".isdigit())
5 print("3.5".isdigit())
```

AUSGABE
```
True
False
True
False
False
```

`"Hallo Welt"` enthält ein Leerzeichen, und das ist kein Buchstabe. `"-5"` und `"3.5"` sind zwar Zahlen, aber das Minuszeichen und der Punkt sind keine Ziffern.

Damit kannst du den `ValueError` vermeiden, der bei `int(...)` mit einer falschen Eingabe entsteht: Das Programm wandelt die Eingabe nur um, wenn sie aus Ziffern besteht.

```
1 eingabe = input("Wie alt bist du? ")
2 if eingabe.isdigit():
3     print("In fünf Jahren bist du", int(eingabe) + 5)
4 else:
5     print("Bitte gib eine ganze Zahl mit Ziffern ein.")
```

AUSGABE
```
Wie alt bist du? fünfzehn
Bitte gib eine ganze Zahl mit Ziffern ein.
```

### 1.10.9 Zusatzwissen: Ausschnitte (Slicing)

**Werkzeug: Slicing**

Diesen Teil brauchst du für die Aufgaben nicht – er zeigt dir eine praktische Abkürzung. Mit zwei Indizes und einem Doppelpunkt in den eckigen Klammern schneidest du einen **Ausschnitt** (englisch *slice*) aus einer Zeichenkette heraus: `text[start:ende]`. Das Zeichen am Index `start` gehört dazu, das Zeichen am Index `ende` **nicht** mehr.

```
1 text = "Informatik"
2 print(text[0:4])
3 print(text[2:6])
4 print(text[::-1])
```

AUSGABE
```
Info
form
kitamrofnI
```

`text[0:4]` enthält die Zeichen mit den Indizes 0, 1, 2 und 3. Die Schreibweise `text[::-1]` ist ein Trick: Sie durchläuft den ganzen Text mit der Schrittweite −1, also rückwärts.

**Merke: Zusatzwissen**

Ein **Ausschnitt** (Slice) wird mit `text[start:ende]` erzeugt (das Zeichen am Index `ende` ist **nicht** mehr enthalten).

### 1.10.10 Übungen

**Arbeitsauftrag 119: Zeichen-Steckbrief**

Schreibe ein Programm, das ein Wort einliest und Folgendes ausgibt:

- die Länge des Wortes,
- das erste und das letzte Zeichen,
- das Wort rückwärts.

So könnte es aussehen:

AUSGABE

```
Wort eingeben: Python
Länge: 6
Erstes Zeichen: P
Letztes Zeichen: n
Rückwärts: nohtyP
```

**Tipp**

Für das Rückwärts-Wort baust du in einer Schleife einen neuen Text auf, wie im Abschnitt *Einen neuen Text aufbauen*. Der Trick: Hänge jedes Zeichen **vorne** an, nicht hinten. Also nicht `rueckwaerts + zeichen`, sondern `zeichen + rueckwaerts`.

**Arbeitsauftrag 120: Vokale zählen**

Schreibe ein Programm, das einen Satz einliest und mit einer Schleife zählt, wie viele Vokale (a, e, i, o, u) darin vorkommen. Gib die Anzahl aus. Auch großgeschriebene Vokale sollen mitgezählt werden.

**Tipp**

Kombiniere drei Dinge aus diesem Abschnitt: die Schleife über alle Zeichen, die Zählvariable aus dem Beispiel `"Anna"` und die Prüfung `zeichen in "aeiou"`. Gegen das Problem mit den Großbuchstaben hilft `.lower()`.

**Arbeitsauftrag 121: Passwort prüfen**

Ein Passwort gilt hier als sicher, wenn es **mindestens 8 Zeichen** lang ist und **mindestens eine Ziffer** enthält. Implementiere eine Funktion `ist_sicher(passwort)`, die `True` oder `False` zurückgibt. Teste sie mit einem zu kurzen Passwort, einem langen Passwort ohne Ziffer und einem sicheren Passwort.

**Arbeitsauftrag 122 (Zusatz): Palindrom**

Ein **Palindrom** ist ein Wort, das vorwärts und rückwärts gelesen gleich ist, z. B. „Anna“ oder „Rentner“. Schreibe eine Funktion `ist_palindrom(wort)`, die `True` oder `False` zurückgibt. Groß- und Kleinschreibung soll dabei keine Rolle spielen (Tipp: `text.lower()`). Teste mit „Anna“, „Rentner“ und „Informatik“.

### 1.10.11 Weitere Übungen

Hier kannst du das Gelernte weiter üben. Die Punkte im Titel zeigen, wie schwer eine Aufgabe ist: ●○○ leicht, ●●○ mittel, ●●● knifflig. Fang mit den leichten an.

**Arbeitsauftrag 123: Index und Zeichen ●○○**

Gegeben ist `wort = "Freibad"`. Bestimme **ohne Computer** die Werte der Ausdrücke. Prüfe danach mit `print(...)` am Computer.

| wort[0] | wort[3] | wort[-1] | len(wort) | wort[len(wort) - 2] |
|---|---|---|---|---|
| | | | | |

Gib außerdem an, an welchem Index das Zeichen `b` steht.

Das Zeichen b steht am Index:

**Arbeitsauftrag 124: Ausgabe vorhersagen ●○○**

Gib **ohne Computer** an, was das Programm ausgibt. Überprüfe danach am Computer.

```
1 text = "Klasse 10a"
2 print(len(text))
3 print(text[7] + text[8])
4 print("s" in text)
5 print("S" in text)
6 print(text.lower())
```

Meine Vorhersage:

### Arbeitsauftrag 125: Initialen ●○○

Implementiere eine Funktion `initialen(vorname, nachname)`, die die Initialen mit Punkten zurückgibt, zum Beispiel für „Mia“ und „Becker“ die Zeichenkette `M. B.`. Teste die Funktion mit zwei verschiedenen Namen.

### Arbeitsauftrag 126: Buchstaben zählen ●○○

Implementiere eine Funktion `zaehle(text, zeichen)`, die zurückgibt, wie oft `zeichen` in `text` vorkommt. Teste sie mit diesen Aufrufen:

| Aufruf | erwartet | tatsächlich |
|---|---|---|
| `zaehle("Banane", "a")` | 2 | |
| `zaehle("Mississippi", "s")` | 4 | |
| `zaehle("Hallo", "x")` | 0 | |

### Arbeitsauftrag 127: Zwei Fehler finden ●●○

Das Programm soll zählen, wie oft der Buchstabe e im Satz vorkommt, egal ob groß oder klein. Erwartet ist 7. Stattdessen bricht es mit einem Fehler ab.

```
satz = "Eine Elefantenherde"
satz.lower()
anzahl = 0
for i in range(1, len(satz) + 1):
    if satz[i] == "e":
        anzahl = anzahl + 1
print(anzahl)
```

1. Gib an, welche Fehlermeldung erscheint, und erkläre sie.

2. Das Programm enthält noch einen zweiten Fehler. Er würde dafür sorgen, dass die großen E nicht mitgezählt werden. Erkläre ihn.
3. Schreibe das Programm richtig auf.

Die beiden Fehler:

**Arbeitsauftrag 128: Ohne Leerzeichen ●●○**

Implementiere eine Funktion `ohne_leerzeichen(text)`, die eine neue Zeichenkette zurückgibt, in der alle Leerzeichen fehlen. Teste mit `"Hallo Welt"` (erwartet: `HalloWelt`) und mit `"0151 234 567"` (erwartet: `0151234567`).

**Tipp**

Baue wie im Abschnitt *Einen neuen Text aufbauen* einen neuen Text auf. Beginne mit `""` und hänge nur die Zeichen an, die **kein** Leerzeichen sind. Ein Leerzeichen schreibst du als `" "`.

**Arbeitsauftrag 129: Vokale tauschen ●●○**

Ein bekanntes Sprachspiel: Man ersetzt in einem Satz alle Vokale durch denselben Vokal. Aus „Wir sehen uns morgen“ wird dann „Wir sihin ins mirgin“. Implementiere eine Funktion `nur_i(text)`, die eine neue Zeichenkette zurückgibt, in der jeder Vokal (a, e, i, o, u, auch großgeschrieben) durch ein kleines `i` ersetzt ist. Alle anderen Zeichen bleiben unverändert. Teste mit `"Wir sehen uns morgen"` und mit `"Otto"`.

**Arbeitsauftrag 130: Handynummer verbergen ●●●**

Auf Rechnungen wird eine Handynummer oft nur teilweise gezeigt, zum Beispiel `*********678`. Implementiere eine Funktion `verberge(nummer)`, die eine neue Zeichenkette zurückgibt. Darin ist jedes Zeichen durch `*` ersetzt – außer den **letzten drei** Zeichen. Teste mit `"015112345678"` (erwartet: `*********678`) und `"12345"` (erwartet: `**345`).

**Tipp**

Hier brauchst du die **Position** jedes Zeichens. Lass die Schleife deshalb über die Indizes laufen: `for i in range(len(nummer)):`. Überlege: Ab welchem Index beginnen die letzten drei Zeichen? Bei 12 Zeichen sind das die Indizes 9, 10 und 11.

### 1.10.12 Das Wichtigste in Kürze

- Eine **Zeichenkette** ist eine **Sequenz von Zeichen**. Jedes Zeichen hat einen **Index**, gezählt ab 0.
- `text[i]` liefert das Zeichen am Index `i`; `text[-1]` ist das letzte Zeichen.
- `len(text)` liefert die Länge. Gültige Indizes reichen von 0 bis `len(text) - 1`, sonst gibt es einen **IndexError**.
- Mit `+` verkettet man Zeichenketten. Ein **f-String** `f"... {wert} ..."` setzt Werte bequem in einen Text ein.
- `for zeichen in text:` durchläuft alle Zeichen, `for i in range(len(text)):` alle Indizes.
- `x in text` prüft, ob `x` enthalten ist. `.upper()` und `.lower()` liefern eine neue Zeichenkette; `.isalpha()` und `.isdigit()` prüfen auf Buchstaben bzw. Ziffern.

**Selbstcheck**

- ☐ Ich kann erklären, warum eine Zeichenkette eine Sequenz von Zeichen ist, und zu jedem Zeichen den Index angeben.
- ☐ Ich kann mit `text[i]` und `len(text)` das erste, ein beliebiges und das letzte Zeichen ausgeben.
- ☐ Ich kann erklären, wann ein `IndexError` entsteht, und ihn vermeiden.
- ☐ Ich kann Zeichenketten verketten und einen f-String schreiben.
- ☐ Ich kann mit einer Schleife alle Zeichen durchlaufen, z. B. um Zeichen zu zählen oder einen Text umzudrehen.
- ☐ Ich kann `in`, `.upper()`, `.lower()`, `.isalpha()` und `.isdigit()` sinnvoll einsetzen.

Gymnasium Ottweiler