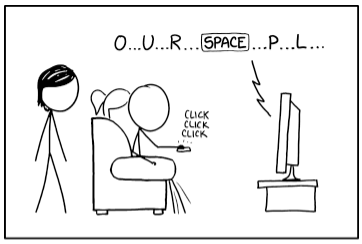
TEXT ENTRY https://xkcd.com/2137/



THE WEIRDEST THING ABOUT 2019 IS OBVIOUSLY THAT DONALD TRUMP IS PRESIDENT, BUT I THINK THE SECOND WEIRDEST IS THAT YOU SOMETIMES *STILL* HAVE TO TYPE STUFF IN BY PICKING LETTERS ON A SCREEN ONE AT A TIME WITH A CURSOR LIKE YOU'RE ENTERING A HIGH SCORE IN A 1980s ARCADE GAME.

I like to think that somewhere out there, there's someone whose personal quest is lobbying TV providers to add an option to switch their on-screen keyboards to Dvorak.

8. Benutzereingabe

Programme arbeiten oft nach dem EVA[1]-Prinzip. Abbildung 8.1 zeigt das Prinzip mit Beispielen. Der `input`-Funktionsaufruf kann in Python dazu eingesetzt werden, dass der **Benutzer** (!) des Programms eine Eingabe (eng. input) an das Programm über die Tastatur durchführen kann. Diesen Funktionsaufruf schauen wir uns in diesem Kapitel genauer an. Die Ausgabe am Bildschirm kennen wir bereits. Dafür können wir einen `print`-Funktionsaufruf verwenden.

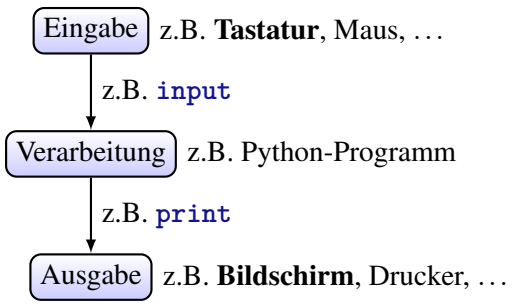


Abbildung 8.1: Die Python-Funktionen `input` realisiert die Eingabe über die Tastatur.

Die Lernziele für dieses Kapitel lauten:

- ☐ Sie erstellen ein Programm, welches eine Benutzereingabe über die Tastatur ermöglicht.
- ☐ Sie benutzen in Python-Programmen formatierten Text.

8.1 Wie funktioniert die `input`-Funktion?

Listing 8.1 zeigt ein Beispiel für den `input`-Funktionsaufruf. Die Benutzereingabe erfolgt über die Tastatur in der **Konsole**.

```
1 vorname = input("Wie lautet Ihr Vorname? ")
2 print(f"Vorname: {vorname}")
```

Listing 8.1: Eingabe und Ausgabe werden miteinander kombiniert (`input_bsp_1.py`).

Der `input`-Funktionsaufruf erlaubt als Argument einen Text. Dieser Text wird **vor** der Benutzereingabe in der Konsole ausgegeben. Damit können wir dem Benutzer des Programms mitteilen, was wir als Eingabe erwarten.

[1] Eingabe-Verarbeitung-Ausgabe

■ **Beispiel 8.1** In Listing 8.1 ist der Text `"Wie lautet Ihr Vorname? "` das Argument des `input`-Funktionsaufrufs. ■

Wichtig! Wenn wir ein Programm mit einem `input`-Funktionsaufruf ausführen, dann wird das Programm in der Zeile mit dem `input`-Funktionsaufruf automatisch **angehalten**. Der Programmablauf wird so lange gestoppt, bis der Benutzer eine Eingabe über die Tastatur in der **Konsole** durchführt und diese mit der Return-Taste („Enter“) abschliesst. ■

Durch die Kombination einer Zuweisung mit einem `input`-Funktionsaufruf, wird die Benutzereingabe in der Variablen gespeichert.

■ **Beispiel 8.2** In Listing 8.1 wird die Eingabe nach dem Drücken der Return-Taste in der Variablen `vorname` gespeichert. ■

Wichtig! Wenn wir mit `input` etwas in der Konsole eintippen, dann benötigen wir für die Eingabe **keine** Anführungszeichen. Python behandelt alles, was mit `input` über die Konsole eingetippt wird, automatisch als **Text**! ■

■ **Beispiel 8.3** Listing 8.2 zeigt eine Ausführung des Programms aus Listing 8.1. Es ist nur ein Beispiel. In diesem Fall hat der Benutzer den Text `Bob` in der Konsole eingegeben. ■

```
Wie lautet Ihr Vorname? Bob
Vorname: Bob
```

Listing 8.2: Beispielausführung für das Programm aus Listing 8.1.

Wir können in einem Programm auch mehrere Benutzereingaben vornehmen. Für jede Benutzereingabe ist ein `input`-Funktionsaufruf notwendig.

■ **Beispiel 8.4** Listing 8.4 zeigt eine Ausführung des Programms aus Listing 8.3. Der Benutzer muss hier zwei Eingaben vornehmen. ■

```
1 vorname = input("Wie lautet Ihr Vorname? ")
2 nachname = input("Wie lautet Ihr Nachname? ")
3 print(f"Vorname: {vorname}")
4 print(f"Nachname: {nachname}")
```

Listing 8.3: Zwei Eingaben in einem Programm (`input_bsp_2.py`).

```
Wie lautet Ihr Vorname? Stanley
Wie lautet Ihr Nachname? Kubrick
Vorname: Stanley
Nachname: Kubrick
```

Listing 8.4: Beispielausführung für das Programm aus Listing 8.3.

8.2 Text formatieren

Wir können innerhalb eines Textes einen Variablennamen notieren. Damit Python während der Ausführung den Variablennamen durch den **gespeicherten Inhalt der Variablen ersetzt**, müssen wir den Variablennamen im Text **markieren**. Listing 8.5 zeigt in Zeile 2, wie wir innerhalb eines Textes die Variable markieren können.

```
1 farbe = input("Farbe? ")
2 print(f"Das Gummibärchen ist {farbe}.")
```

Listing 8.5: Formatierter Text (`f-text_bsp_1.py`).

Definition 8 — Formatierter Text. Benutzen wir **innerhalb** eines Textes die geschweiften Klammern ({ und }) und notieren wir **vor** den doppelten Anführungszeichen den Buchstaben `f`, dann handelt es sich um formatierten Text. Die Variable innerhalb der geschweiften Klammen wird während der Ausführung durch den gespeicherten Inhalt ersetzt.

■ **Beispiel 8.5** Listing 8.6 zeigt die Ausgabe in der Konsole, wenn wir das Programm aus Listing 8.5 ausführen. Die geschweiften Klammern erscheinen **nicht** in der Ausgabe. ■

```
Farbe? rot
Das Gummibärchen ist rot.
```

Listing 8.6: Der Variablennamen wird durch den gespeicherten Inhalt ersetzt.

Wir können natürlich auch mehrere geschweifte Klammern in einem Text verwenden. Listing 8.7 zeigt ein Beispiel mit zwei Textformatierungen (pro Formatierung ein Klammerpaar).

```
1 vorname = input("Wie lautet Ihr Vorname? ")
2 nachname = input("Wie lautet Ihr Nachname? ")
3 print(f"Vorname: {vorname}, Nachname: {nachname}")
```

Listing 8.7: Beide Variablennamen werden während der Ausführung durch den gespeicherten Inhalt ersetzt. (`f-text_bsp_2.py`).

Wichtig! Das Formatieren von Text hat in Python **nichts** mit der Formatierung von Text in einem Textverarbeitungsprogramm (wie in Microsoft Word) zu tun. Sie können in Python keinen Text fett gedruckt darstellen oder unterstreichen. ■

Die Textformatierung ist **nicht** an den `print`-Funktionsaufruf gebunden. Wir können Text an beliebigen Stellen formatieren. Listing 8.8 zeigt ein Beispiel mit einer Zuweisung.

```
1 vorname = input("Wie lautet Ihr Vorname? ")
2 nachname = input("Wie lautet Ihr Nachname? ")
3 nachricht = f"Vorname: {vorname}, Nachname: {nachname}"
4 print(nachricht)
```

Listing 8.8: Formatierter Text in Kombination mit einer Zuweisung (`f-text_bsp_3.py`).

Es gibt zwei häufige **Fehler** im Umgang mit formatiertem Text:

- Der Buchstabe `f` wird vor dem doppelten Anführungszeichen vergessen.
- Die geschweiften Klammern werden gar nicht oder nicht korrekt notiert.

Häufig erkennt man dies an der abweichenden, farblichen Darstellung des Textes in der Python-Datei. Auch die Ausgabe in der Konsole ist dann nicht so wie gewünscht. Listing 8.9 zeigt ein Programm ohne den Buchstaben `f`. Listing 8.10 ein Beispiel für eine Ausgabe in der Konsole, wenn der Buchstabe `f` fehlt.

```
1 farbe = input("Farbe? ")
2 print("Das Gummibärchen ist {farbe}.")
```

Listing 8.9: Der Text wird so **nicht** korrekt formatiert. (`f-text_bsp_4.py`).

```
Farbe? rot
Das Gummibärchen ist {farbe}.
```

Listing 8.10: Die Ausgabe entspricht 1 : 1 dem Text, jedes Zeichen wird ausgegeben.

8.3 Aufgaben

In den folgenden Aufgaben setzen Sie sich mit der `input`-Funktion, Variablen und formatiertem Text auseinander.

8.3.1 Aufgabe 1

Notieren Sie ein Programm, mit **einer** Benutzereingabe. Fragen Sie den Benutzer nach dem Lieblingsschulfach. Geben Sie es dann in der Konsole aus.

8.3.2 Aufgabe 2

Betrachten Sie Listing 8.11. Notieren Sie ein Programm, welches diese Eingabe und Ausgabe erzeugen kann. Verwenden Sie eine Benutzereingabe.

```
Wie lautet Ihr Geburtsdatum? 21.02.1988
Sie haben am 21.02.1988 Geburtstag.
```

Listing 8.11: Eingabe und Ausgabe in der Konsole.

8.3.3 Aufgabe 3

Notieren Sie ein Programm, mit **einer** Benutzereingabe (siehe Listing 8.12). Fragen Sie den Benutzer nach der **Stiftfarbe**. Die Turtle soll dann ein **gleichseitiges** Dreieck mit der Seitenlänge 200 (siehe Abbildung 8.2) zeichnen. Die Ausgabe in der Konsole soll erst dann erfolgen, wenn die Turtle tatsächlich die Stiftfarbe angepasst hat.

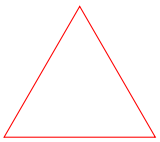


Abbildung 8.2: Gleichseitiges Dreieck.

```
Stiftfarbe? red
Stiftfarbe ist nun red.
```

Listing 8.12: Eingabe und Ausgabe in der Konsole.