Kapitel 3: Grundlagen der Programmierung

3.1. Kommentare

Allem vorangestellt ein Hinweis für das Arbeiten in Python bzw. dem Programmieren im Allgemeinen: Zu gegebener Zeit wird es notwendig sein, seinen Code zu kommentieren. Kommentare werden bei der Ausführung des Programms **ignoriert und nicht ausgeführt**. Diese sind nur für die Programmiererin oder den Programmierer sichtbar. In Kommentaren können wir unser Programm mit zusätzlichen Informationen, Gedanken, Erläuterungen, ... versehen, um so die Nachvollziehbarkeit - auch nach langer Zeit - zu gewährleisten.

Die Syntax ist hierbei wie folgt:

In [ ]:
```
# ein Kommentar
```

Dass es sich tatsächlich um einen Kommentar handelt, erkennt man auch am anderen **Syntax-Highlighting**.

3.2. Syntax-Highlighting

Syntax-Highlighting ist eine Funktion von Texteditoren oder Entwicklungsumgebungen (IDEs, zum Beispiel: VS-Code), die Teile des Quellcodes farblich hervorhebt, um dessen Lesbarkeit zu verbessern und die Struktur nachvollziehbarer zu machen.

Verschiedene Bestandteile des Codes werden dabei in unterschiedlichen Farben oder Stilen hervorgehoben, um die verschiedenen syntaktischen Elemente zu unterscheiden. Dies erleichtert die Arbeit beim Programmieren ungemein und kann dabei helfen, Fehler schneller zu erkennen.

Typischerweise hebt Syntax-Highlighting folgende Elemente hervor:

- **Schlüsselwörter (Keywords):** Diese sind Teil der Programmiersprache, wie `if`, `else`, `while`, etc. Sie werden oft in einer Farbe, wie Blau oder Rot, hervorgehoben.
- **Kommentare:** Kommentare im Code, die von der Programmiersprache ignoriert werden, aber für den Entwickler nützlich sind, werden oft in einer anderen Farbe (z. B. Grau oder Grün) angezeigt.
- **Zeichenketten (Strings):** Text, der in Anführungszeichen eingeschlossen ist (wie `'Hello World'`), wird normalerweise in einer anderen Farbe, wie Gelb oder Rot, dargestellt.
- **Funktionen und Methoden:** Die Namen von Funktionen und Methoden können ebenfalls hervorgehoben werden, um sie von Variablen und anderen Codeelementen zu unterscheiden.
- **Variablen:** Manche Editoren markieren Variablen in einer eigenen Farbe, um sie von Funktionen oder Schlüsselwörtern zu unterscheiden.
- **Zahlen und Symbole:** Zahlen und Operatoren wie `+`, `-`, `*`, etc., werden oft ebenfalls in unterschiedlichen Farben dargestellt.

Zusammengefasst: Syntax-Highlighting erleichtert Entwicklern das Verstehen des Codes, indem es die visuelle Unterscheidung von Syntaxelementen ermöglicht.

3.3. Eingabe, Verarbeitung, Ausgabe und Speicherung (EVAS-Prinzip)

Variablen

Variablen sind eines der wichtigsten Elemente in der Programmierung: Sie dienen dem **Speichern von Werten**.

Vorstellen kann man sich eine Variable als einen kleinen Bereich im Speicher, dem ein Name gegeben wurde. Dieser Speicherplatz wird nur für diese Variable reserviert. In den meisten höheren Programmiersprachen wird zwischen der **Deklaration** (dem Anlegen der Variablen) und der **Initialisierung** (dem ersten Belegen mit einem Wert) einer Variable unterschieden. In Python existieren Variablen nur durch **Zuweisung**, also Initialisierung. Deshalb spricht man davon, dass eine Variable durch die erste Zuweisung gleichzeitig **deklariert und initialisiert** wird.

Der nachfolgende Code-Block zeigt eine solche Zuweisung. Die Variable mit dem Namen `ich_bin_eine_variable` wird deklariert und im gleichen Atemzug mit dem Wert `42` initialisiert.

In [ ]:
```
ich_bin_eine_variable = 42
```

Aber wie genau funktioniert eine **Zuweisung**?

Variablenzuweisungen folgen diesem Schema:

```
Variable = Ausdruck
<<<----------------
```

Rechts vom Gleichheitszeichen steht ein Ausdruck. Dieser wird ausgewertet und zu einem Wert. Links vom Gleichheitszeichen steht der Name der Variable. Die Zuweisung erfolgt also immer von rechts nach links - **nicht umgekehrt**! Der Ausdruck auf der rechten Seite kann im einfachsten Fall eine Zahl, eine Zeichenkette oder wiederum eine Variable sein. Später werden wir auch komplexere Ausdrücke kennenlernen.

Hinweis: Speicherplatz kostet heutzutage so gut wie kein Geld mehr. Daher sind Variablennamen so zu wählen, dass einem unbeteiligten Dritten klar wird, was darin gespeichert ist. Man sollte also nicht an der falschen Stelle sparen. Gute Lesbarkeit von Code zeichnet eine guten Programmiererin oder einen Programmierer im Besonderen aus!

Beispiel für einen schlechten Variablennamen: `sn = 'Mustermann'`

Beispiel für einen guten Variablennamen: `schueler_nachname = 'Mustermann'`

Entsprechend der Python Enhancement Proposals (kurz: PEPs) - quasi Vorgaben zur Programmierung mit Python - ist im PEP8 vorgeschrieben, wie Variablen benannt werden sollten. Dabei wird der **Snake Case** als Namenskonvention festgelegt. Dabei werden alle Worte in Kleinbuchstaben geschrieben (engl.: lowercase) und alle potenztiellen Leerzeichen durch Unterstriche `_` ersetzt (engl.: underscores). Sonderzeichen **dürfen nicht** verwendet werden. Variablennamen **dürfen nicht** mit Zahlen beginnen.

Eine kleine Gedankenstütze zur Namenskonvention ist die Programmiersprache selbst. Diese heißt `Python`. Und Pythons sind Schlangen. Also: **Snake Case**.

Bei der Programmierung - in zum Beispiel anderen Programmiersprachen - gibt es auch andere Namenskonventionen. Zum Beispiel:

In [ ]:
```
schuelerNachname = 'Mustermann'
```

Auffällig an diesem Beispiel ist, dass das `N` von `Nachname` groß geschrieben ist. Diese Schreibweise nennt man **`camelCase`**. Dabei handelt es sich um eine Praxis des Schreibens von Phrasen ohne Leerzeichen, Trennzeichen oder anderen Interpunktionen. Gängige Beispiele kennt ihr aus dem Alltag. Zum Beispiel: "YouTube", "iPhone" und "eBay". Camel Case wird oft als Namenskonvention in der Programmierung (Java, JavaScript, PHP, ...) verwendet. Es wird auch manchmal in Online-Benutzernamen wie "JohnSmith" verwendet oder um mehrwortige Domainnamen lesbarer zu machen, zum Beispiel bei der Werbung für "EasyWidgetCompany.com".

Im Nachfolgenden halten wir uns konsequent an die **Snake Case**-Schreibweise.

Datentypen

Eine Variable speichert Werte (auch: Literals). Bei diesen zu speichernden Werten unterscheiden wir **Datentypen**. Ein Datentyp ist ein Schema für die Speicherung von Daten. Dabei wird festgelegt, wie und in welcher Art und Weise die Bits gespeichert werden.

Wir unterscheiden zunächst nur die nachfolgenden *primitiven* Datentypen:

Bezeichnung	Kurzform	Bedeutung	Beispiel(e)	Hinweis
integer	int	ganze Zahlen	`3`	-
float	float	Dezimalzahl bzw. Gleitkommazahl	`3.1`, `4.2`	Kommas werden in **ENGLISCHER** Notation als **Dezimalpunkt** geschrieben!
string	str	Zeichenkette	`'Hallo'`, `'Test'`, `'Test123'`	Zeichenketten werden mit einfachen '' oder doppelten Hochkommas "" angegeben. Einzelne Hochkommas sind dabei der Standard! Zahlen in doppelten oder einfachen Hochkommas werden als Zeichenketten interpretiert. Die Hochkommas dürfen **nicht gemischt** verwendet werden.
boolean	bool	boolesche Werte / Wahrheitswerte	`True` oder `False`	Beide Worte werden jeweils mit einem großen Anfangsbuchstaben geschrieben!
none	none	der "Nichts"-Datentyp	`None`	Auch hier wird der Wert mit einem großen Anfangsbuchstaben geschrieben!

Möchte man herausfinden, welchen Datentyp eine Variable speichert, so kann man die Pythonfunktion `type(variable)` verwenden. Nachfolgend findet ihr zwei Beispiele. Klickt in die Code-Blöcke hinein und führt diese über den Play-Button aus. Es wird eine Ausgabe erzeugt.

In [ ]:
```
super_tolle_variable = 420.187
type(super_tolle_variable)
```

In [ ]:
```
auch_eine_sehr_gute_variable = 'BvC'
type(auch_eine_sehr_gute_variable)
```

Ausgaben

Damit ein Python-Programm etwas in der Konsole ausgibt, wird der Befehl `print()` verwendet.

In [ ]:
```
test = 'Ich bin ein Test'
print(test)
print('--- (hoffentlich) erfolgreich ---')
```

Dabei kann nicht nur eine einzige Variable ausgegeben werden, sondern auch beliebig viele Texte und Variablen. Variante eins ist dabei das Trennen durch Kommas.

In [ ]:
```
variable_eins = 'No'
variable_zwei = 'mercy'
print(variable_eins, 'Backup,', 'no', variable_zwei, '.')
```

Es fällt auf, dass nach jedem Argument automatisch ein Leerzeichen in der Ausgabe erzeugt wird. Dadurch erscheint zwischen „mercy“ und dem abschließenden Punkt ein (unerwünschtes) Leerzeichen.

Zur besseren Verbindung von Variablen und Text - ohne Leerzeichen-Probleme - kann der sogenannte `F-String` genutzt werden. Dies ist eine Notation, die es ermöglicht, Text und Variablen elegant und nachvollziehbar bei der Ausgabe miteinander zu verbinden. Dabei wird der Ausgabe ein kleines `f` vorangestellt und die einzufügende Variable in **geschweifte Klammern** geschrieben.

In [ ]:
```
ergebnis = 42
print(f'Die Antwort auf die Frage nach dem Leben, dem Universum und dem ganzen Rest
```

Eingabe

Damit Nutzerinnen und Nutzer auch Eingaben tätigen können, steht in Python der `input()` -Befehl zur Verfügung. Wenn ihr den Code ausführt, öffnet sich im oberen Bereich von Visual Studio Code ein Eingabefenster. Dort könnt ihr die Eingabe tätigen. Bestätigt eure Eingabe mit der Enter-Taste.

In [ ]:
```
eingabe = input('Bitte gib deinen Namen ein: ')
print(f'{eingabe}? Toller Name! Viel Spaß beim Programmieren lernen! :-)')
```

Dabei ist zu beachten, dass eine Eingabe **IMMER** und **AUSSCHLIESSLICH** vom Typ `string` ist. Sollte man beispielsweise mit den Eingaben der Nutzerinnen und Nutzer rechnen wollen, **muss** in den benötigten Datentyp konvertiert werden. Dazu stehen in Python ebenfalls entsprechende Funktionen bereit. Mehr dazu im folgenden Abschnitt.

Konvertierung von Datentypen (oder "Typisierung")

Mit Zeichenketten kann man im Allgemeinen nur schlecht rechnen. Soll die Nutzerin oder der Nutzer eine Zahl eingeben, steht man erst einmal vor einem Problem(chen). Die nachfolgenden Beispiele zeigen aber, wie man den Datentyp einer Variable verändern kann.

Konvertierung zu einer Zeichenkette:

In [ ]:
```
variable = 5
string_variable = str(variable)
print(string_variable, type(string_variable))
```

Konvertierung zu einer ganzen Zahl:

In [ ]:
```
variable = '2'
int_variable = int(variable)
print(int_variable, type(int_variable))
```

Konvertierung zu einer Gleitkommazahl:

In [ ]:
```
variable = '1.1'
float_variable = float(variable)
print(float_variable, type(float_variable))
```

Konvertierung zu einem Boolesches Wert / Wahrheitswert:

In [ ]:
```
variable = '1'
bool_variable = bool(variable)
print(bool_variable, type(bool_variable))
```

Nutzt man in der Ausgabe den Datentyp `float` will das Ganze mittels + an einen String anfügen, muss man den Float zuerst explizit mittels `str()` in einen String umwandeln.

Nutzt man das Komma, ist dies unproblematisch.

In [ ]:
```
a = 1.2

print('Ausgabe: ', a) # kein Fehler
print('Ausgabe: ' + str(a)) # kein Fehler
print('Ausgabe: ' + a) # Fehler
```

Rechenoperationen und Operatoren

Operator	Bezeichnung	Beispiel	Hinweis
`+`	Addition	`zahl = a + b`	Bei Zeichenketten bewirkt die "Addition" eine Verbindung zweier Zeichenketten, so würde aus `'Teil1' + 'Teil2'` ein zusammengesetzter String `'Teil1Teil2'` entstehen.
`-`	Subtraktion	`ergebnis = a - b`	-
`*`	Multiplikation	`ergebnis = a * b`	Bei Zeichenketten sorgt eine Multiplikation mit einer ganzen Zahl dafür, dass ein Vielfaches der Zeichenkette erzeugt wird. Zum Beispiel würde `'a' * 4` den String `aaaa` erzeugen.
`/`	Division	`ergebnis = a / b`	Das Ergebnis einer Division ist immer vom Typ `float`.
`//`	ganzzahlige Division	`ergebnis = a // b`	Gibt als Ergebnis zurück, wie oft sich eine Zahl ganzzahlig teilen lässt.
`%`	Modulo / Restklassendivison	`ergebnis = a % b`	Gibt den verbleibenden Rest bei ganzzahliger Divison zurück.
`**`	Potenz	`quadrat = a**2`	In diesem Beispiel wird die Quadratzahl berechnet. Alle anderen Exponenten sind ebenfalls denkbar.
`or`	Logisches ODER	`a or b`	Vergleicht zwei boolesche Werte. Wenn einer von beiden den Wahrheitswert `True` hat, gibt or ebenfalls `True` zurück.
`and`	Logisches UND	`a and b`	Vergleicht zwei boolesche Werte. Beide Werte müssen den Wahrheitswert `True` haben, sonst gibt and `False` zurück.
`not`	Logisches NICHT	`not a`	Kehrt den booleschen Wert um. Aus einem `True` wird ein `False` und umgekehrt.
`<`	kleiner als	`a < b`	Gibt `True` zurück, wenn der Vergleich zutrifft, sonst `False`.
`<=`	kleiner gleich	`a <= b`	Gibt `True` zurück, wenn der Vergleich zutrifft, sonst `False`.

Operator	Bezeichnung	Beispiel	Hinweis
`>`	größer	`a > b`	Gibt `True` zurück, wenn der Vergleich zutrifft, sonst `False`.
`>=`	größer gleich	`a >= b`	Gibt `True` zurück, wenn der Vergleich zutrifft, sonst `False`.
`!=`	ungleich	`a != b`	Gibt `True` zurück, wenn die Werte verschieden sind, sonst `False`.
`==`	gleich	`a == b`	Gibt `True` zurück, wenn die Werte gleich sind, sonst `False`.

⚠️ Hinweis zum Schreibstil ⚠️

Es ist üblich, zwischen Rechenoperationen, Operatoren, Variablen und dem Gleichheitszeichen **IMMER** ein Leerzeichen zu setzen. In obiger Tabelle und in allen vorangegangenen Beispielen wird dies konsequent umgesetzt.

Das erhöht die Lesbarkeit deutlich und erleichtert das Verständnis des Programmcodes.