Textbasiert programmieren

Um genauer zu verstehen, wie der OzoBot über die Blöcke tatsächlich gesteuert wird, brauchen wir einen kleinen Exkurs in das Programmieren mit Python. Zudem werden dir komplexere Programme mit den Blöcken viel besser gelingen, wenn du mehr über die Hintergründe und die Bedeutung der einzelnen Elemente weißt. Wir konzentrieren uns im ersten Schritt auf Bedingungen und Schleifen, müssen aber auch schon Variablen einführen.

Damit wir keine Software auf dem Rechner installieren müssen, nutzen wir zum Programmieren eine Online-Umgebung. Falls die eine nicht funktioniert, nutzt du einfach die andere.

1. https://www.online-python.com/
2. https://www.programiz.com/python-programming/online-compiler/

Dort hinein kannst du deinen Code schreiben, laufen lassen und auch herunterladen.

Variablen

Variablen können Werte speichern. Das ist sehr praktisch, wenn man Werte in einem Programm immer wieder einmal braucht. Schau dir hier an, wie Variablen in Python umgesetzt werden.

Für die ganz Schnellen unter euch, die eher an Beispielen lernen:

variablen.py

```
# Variablen einen Wert zuweisen
wahrheitswert = True
zahl = 42
pi = 3.14
vorname = 'Axel'

# Variablen ausgeben
print(wahrheitswert)
print(vorname)
print(zahl)
print(pi)
```

Variablen setzt Python automatisch auf den richtigen Datentyp, d.h. Python erkennt an eurer Zuweisung, ob es sich um eine boolesche Variable (wahrheitswert), einen String (Zeichenkette, vorname), einen Integer (ganze Zahl, zahl) oder Gleitkommazahl (Zahl mit Komma, pi) handelt. Ungewöhnlich ist die boolsche Variable für dich. Einer solchen Variablen kannst du die Werte True/False zuweisen (Großschreibung ist wichtig!). Später dazu mehr.

Wie habe ich das mit der Anzeige gemacht?

Du hast bestimmt gesehen, dass der Code dieses ersten Programmes hier im

Wiki gut dargestellt wird, und du dir das Programm durch Klick auf „variablen.py“ herunterladen kannst. Das geht hier im Wiki ganz einfach:

```
<file python variablen.py>
# Variablen einen Wert zuweisen
wahrheitswert = True
zahl = 42
pi = 3.14
vorname = 'Axel'

# Variablen ausgeben
print(wahrheitswert)
print(vorname)
print(zahl)
print(pi)
</file>
```

„python“ gibt an, dass es sich um Python-Code handelt, „variablen.py“ ist einfach der Dateinname, der zum Download angeboten wird.

Bedingungen

Ich hatte dir schon bei den Blöcken die Entsprechung in Python-Code angegeben. Ausführlich wird das Ganze nochmal hier erklärt.

Für die ganz Schnellen unter euch, die eher an Beispielen lernen:

Einfache Bedingungen

bedingung.py

```
# Hier wird ein Wert zugewiesen (einfaches Gleichheitszeichen)
zahl = 42

# Hier wird auf Gleichheit überprüft (zweifaches Gleichheitszeichen)
if wert == 42:
    print('Das ist die Zahl der Zahlen!')

print('und hier geht es nach der if-Abfrage weiter')
```

Zum if-Block gehören die Zeilen, die gleich eingerückt sind. Das wäre in deinen Blockprogrammen all das, was du in die Lücke setzt.

Bedingungen mit else

bedingung_mit_else.py

```
wert = 9
if wert < 5:
    print('Wert ist kleiner als 5')
else:
    print('Wert ist größer als 4')
```

Bedingungen mit Mehrfachprüfung

Du solltest „elif“ als Kurzschreibweise von „else if“ lesen.

bedingung_mit_elif.py

```
wert = 9
if wert < 5:
    print('Wert ist kleiner als 5')
elif wert == 5:
    print('Wert ist exakt 5')
else:
    print('Wert ist größer als 5')
```

Operatoren bei Bedingungen

- **==** gleich
- **!=** ungleich (!:not, =:equal)
- **<** kleiner
- **>** größer
- **⇐** kleiner gleich
- **>=** größer gleich

Boolesche Vergleiche

- **and** (und)
- **or** (oder)

Hierzu ein Beispiel:

bedingung_mit_boole.py

```
wert = 9
# Wenn der Wert kleiner 15 und größer als 6 ist, dann ... sonst ...
```

```
if wert < 15 and wert > 6:
    print('Wert liegt im Bereich.')
else:
    print('Wert liegt nicht im Bereich.')
```

Schleifen

Die While-Schleife prüft zuerst, ob eine Bedingung zutrifft. Hier gibt es die lange Version. Auch hier muss man wie beim if das einrücken, was zu der Schleife gehört.

while.py

```
durchgang = 1
while durchgang < 11:
    print(durchgang)
    # Wir erhöhen bei jedem Schleifendurchgang die Variable durchgang
um 1 (auf schlau: Wir inkrementieren die Variable)
    durchgang = durchgang + 1
print("nach der Schleife")
```

Nutzereingaben

Manchmal soll der Benutzer einen Wert eingeben. Das ist in Python sehr einfach:

input.py

```
# Die Eingabe wird der Variablen name zugewiesen
name = input("Gib deinen Namen ein: ")
print(name)
```

Typen von Variablen

Standardmäßig liest Python mit der Funktion „input“ eine Zeichenkette ein. Du kannst dir das wie ein Wort in einer Sprache vorstellen. Das Problem: Mit Worten kann man nicht rechnen! Für Python ist z.B. dann auch eine Zahl, die von Input gelesen wird, ein „Wort“.

Python kennt verschiedene Typen von Variablen:

- **str** oder auch „String“ (Zeichenkette, z.B. „Hannes“, „yD&7G“, „123R“)
- **int** oder auch „Integer“ (ganze positive oder negative Zahl, z.B. „-10“, „6678745677556“)

- **float** oder auch Fließkommazahl (Kommazahlen, z.B. „8.998“, „-2.6“)

Wenn du mit Eingaben aus input() rechnen willst, musst du diese Eingabe in den benötigten Datentyp umwandeln („casten“). Das kannst du direkt bei der Eingabe tun:

cast_01.py

```
# Die Eingabe wird der Variablen name zugewiesen
# durch int() wird die Eingabe in eine Ganzzahl
umgewandelt
zahl = int(input("Gib eine ganze Zahl ein: "))
print(zahl)
```

Es geht aber auch nachträglich:

cast_02.py

```
# Die Eingabe wird der Variablen name zugewiesen
zahl = input("Gib eine ganze Zahl ein: ")
# jetzt umwandeln
zahl = int(zahl)
print(zahl)
```

Du musst also ggf. den Datentyp ändern, wenn du mit den Eingabewerten rechnen möchtest!

Aufgaben

Du kannst alle Aufgaben online erledigen: https://www.online-python.com/ oder https://www.programiz.com/python-programming/online-compiler/. Dokumentiere deine Programme auf deiner Wikiseite, so wie ich es weiter oben gezeigt habe.

Schreibe ein Python-Programm, das

1. die Eingabe einer Zahl erwartet und diese Zahl um 10 erhöht ausgibt (zahl = zahl + 10)
2. die Eingabe eines Strings (Zeichenkette) erwartet, diesen String in einer anderen Variable speichert und diese andere Variable ausgibt.
3. zwei Variablen mit Strings belegt, eine neue Variable aus der Summe der Strings bildet und ausgibt.
4. die ersten 10 Werte einer linearen Funktion y = bx + a ausgibt, wobei der Nutzer b und a eingeben können soll.
5. ein Zahlenratespiel durchführt. Du gibst im Programm einen Wert zwischen 0 und 100 vor, den der Nutzer raten soll. Rät er zu klein, gibt es eine Ausgabe „zu klein“ und eine neue Eingabe, ansonsten eine Ausgabe „zu groß“ und eine neue Eingabe, bis die Zahl geraten wurde.

Erweiterung zu Aufgabe 5

Du kannst eine zufällige Zahl zwischen 1 und 100 zu Anfang des Programmes so erzeugen:

zufall.py

```
import random
zufall = random.randint(1,100))
```

... dann wird es auch für dich beim Testen ein wenig spannender.

Lösungen

Aufgabe 1

zahlplus10.py

```
zahl = input("Gib eine Zahl ein: ")
zahl = int(zahl)
print("Das Ergebnis lautet: ")
print(zahl+10)
```

Aufgabe 2

copytovar.py

```
eingabe = input("Gib etwas ein: ")
neu = eingabe
print("Ausgabe: ")
print(neu)
```

Aufgabe 3

combinestrings.py

```
teil01 = "Haus"
teil02 = "tier"
kombi = teil01+teil02
```

```
print(kombi)
```

Aufgabe 4

linearfunc.py

```
b = input("Gib die Steigung ein: ")
a = input("Gib den Offset ein: ")
b = int(b)
a = int(a)
x = 0
while x < 10:
    print(x*b+a)
    x = x + 1
```

Codeanalysen

Was macht dieses „Programm“ und welchem Block bei OzoBlockly entspricht es?

analyse1.py

```
schalter = True

while schalter
   print('ewig ...')
```

Welchen Wert bekommt „eingabe“, wenn nichts eingegeben wird? Was „prüft“ die if-Bedingung eigentlich?

analyse2.py

```
eingabe = input('Gib etwas ein: ')

if eingabe:
   print(eingabe)
else:
   print('Faules Kind!')
```

Arrays

Arrays sind spezielle Datenstrukturen, die mehrere Variablen enthalten können. Man nennt diese

Variablen hier Elemente. Auf die einzelnen Elemente wird mit einem Index zugegriffen. Man beginnt bei der Zählung immer mit null. Arrays können alle Datentypen enthalten, jedoch sollten alle Elemente eines Arrays zunächst immer vom gleichen Typ sein.

Strenggenommen gibt es in Python eigentlich keine Arrays, sondern man kann Listen so wie Arrays verwenden, aber das ist erstmal eine Spitzfindigkeit.

array_00.py

```
autos = ["vw", "bmw", "toyota", "ford"]
noten = [1,2,3,4,5,6]
print(autos[0])
print(noten[2])
```

Wichtige Arrayfunktionen

Oft braucht man die Länge eines Arrays. Das geht über die Funktion len(). Die Länge gibt die Anzahl der Elemente eines Arrays an.

array_01.py

```
autos = ["vw", "bmw", "toyota", "ford"]
anzahl_autos = len(autos)
print(anzahl_autos)
```

Manchmal muss man ein leeres Array mit einer vorgebenen Anzahl an Elementen vorbereiten (initialisieren).

array_02.py

```
autos = [0]*10
```

... erzeugt an Array aus 10 Elementen, die alle den Wert 0 besitzen.

Die For-Schleife

Fast speziell für ein Array konzipiert ist die For-Schleife. Mit dieser kann man sehr bequem durch ein Array durchlaufen. Das geschieht in folgendem Code. Das Array „autos“ hat vier Elemente - len(autos).

array_02.py

```
autos = ["vw", "bmw", "toyota", "ford"]
anzahl_autos = len(autos)

for i in range(anzahl_autos):
    print(autos[i])
```

range() gibt (für uns erstmal) an, wie oft die For-Schleife durchlaufen werden soll - in diesem Beispiel viermal. i ist die sogenannte „Laufvariable“, die bei jedem Schleifendurchlauf beginnend mit Null um 1 erhöht wird (i=i+1).

Strenggenommen handelt es sich bei der Pythonversion der For-Schleife eigentlich um eine foreach-Schleife. Auch das ist eine Spitzfindigkeit.

Wir können in range() auch einen Anfangswert übergeben:

array_03.py

```
autos = ["vw", "bmw", "toyota", "ford"]
anzahl_autos = len(autos)

for i in range(2,4):
    print(autos[i])
```

Der letzte Code gibt nur die Array-Elemente 3 (autos[2]) und 4 (autos[3]) aus.

Aufgaben

Aufgabe 1

Schreibe ein Programm, welches alle Elemente (mindestens 10) eines Arrays aus Integern aufsummiert.

loesung01.py

```
notenspiegel = [1,2,2,1,1,3,1,2,3,4,1,2,1,1,1,1]
num_students = 0

for i in range(16):
    num_students = num_students + notenspiegel[i]
```

Aufgabe 2

Eine Klausuraufgabe lautete: „Schreibe ein Python-Programm, das den Durchschnitt des folgenden Punktespiegels einer Informatikklausur berechnet. Weiterhin soll der Prozentanteil der Klausuren unter 5 Punkten berechnet werden. Du darfst dazu nichts im Kopf addieren.“

Mohamed hat die Aufgabe als einziger richtig gelöst - allerdings ist der Code überhaupt nicht schön, weil ihm noch keine Arrays zur Verfügung standen.

mohamed.py

```
l =
1*0+2*1+2*2+1*3+1*4+3*5+1*6+2*7+3*8+4*9+1*10+2*11+1*12+1*13+1*14+1*15
m = 1+2+2+1+1+3+1+2+3+4+1+2+1+1+1+1
n = str(l/m)
k = 1*0+2*1+2*2+1*3+1*4+3*5+1
p = 1+2+2+1+1+3+1
q = "Der Durchschnitt beträgt:" + n
h = str(k/p)
print(q)
print(h)
```

Löse die Aufgabe, indem du Arrays verwendest. Nutze einmal dafür die dir schon bekannte While-Schleife und einmal die neue For-Schleife, schreibe also zwei Programme.

Tipp: Speichere die Anzahl der Klausuren in einem Array mit 16 Elementen. Der Index repräsentiert dann einfach die Anzahl der Punkte. Welche weiteren Vorteile hat das Verfahren mit den Arrays für die Lehrkraft im Vergleich zu Mohameds Lösung?

loesung02.py

```
notenspiegel = [1,2,2,1,1,3,1,2,3,4,1,2,1,1,1,1]
num_students = 0
num_students_failed = 0
sum_grades = 0

for i in range(16):
    num_students = num_students + notenspiegel[i]
    sum_grades = sum_grades + (notenspiegel[i] * i)

for i in range(4):
    num_students_failed = num_students_failed + notenspiegel[i]

average = round(float(sum_grades/num_students),1)
failed = round(float(num_students_failed/num_students)*100,1)

print("Durchschnitt: ",average)
print("Unter Schnitt: ",failed,"%")
```