Schulung Blinkenlights

Komplett ohne irgendwas Blinkendes

Sebastian Köll, Jan Rüther

August 14, 2020

1 Aufgabe 1: Python zum Fliegen bringen

1.1 Aufgabe 1a

Bringt das folgende Programm zum laufen und testet es

```
import math


def main():

    radius = int(input("Kugelradius in Metern angeben;"))

    oberflaeche = 4 * math.pi * radius**2
    volumen = (4/3) * math.pi * radius**3

    print(f" Die Obefleache der Kugel betraegt = {oberflaeche}
                                        Quadratmeter")
    print(f" Das Volumen der Kugel betraegt = {volumen} Kubikmeter"
                                        )


if __name__ == '__main__':
    main()
```

1.2 Aufgabe 1b

Schreibt ein Programm (ähnlich dem in Aufgabe 1a), welches die Höhe h und den Radius r eines Zylinders einliest und dann die gesamte Oberfläche und das Volumen des Zylinders berechnet.

1.3 Aufgabe 1c

Schreibt ein Programm, welches für eine positive ganze Zahl die Quersumme berechnet. Die Quersumme von 1337 bespielsweise ist 14.

2 Aufgabe 2: Dinge mit Listen und Schleifen

Schreibt ein Programm, das eine Folge von Komma-Zahlen einliest. Das Ende der Zahlenfolge wird erkannt durch das erste Zeichen, welches keine Zahl (also z.B. ein Buchstabe)ist. Das Programm soll dann für die eingelesenen Zahlen folgende Größen berechnen:

a. Die Anzahl der eingelesenen Zahlen,

b. Die Summe der eingelesenen Zahlen,

c. Das Maximum der eingelesenen Zahlen,

d. Das Minimum der eingelesenen Zahlen,

e. Den Mittelwert der eingelesenen Zahlen

3 Struktogramme

Beim Programmieren ist es häufig hilfreich, sich vorab die Zeit zu nehmen und das Programm durchzuplanen, bevor mit der eigentlichen Programmierung angefangen wird. Um Programme zu planen existiert eine Vielzahl an grafischen Modellierungshilfen. Diese sind dazu da, Programmaufbau und Ablauf in grafischer Form übersichtlich darzustellen. Eine Möglichkeit hierfür sind so genannte Struktogramme (auch Nassi-Schneiderman Diagramme).
Das Diagramm besteht aus verschiedenen Elementen, welche die Elemente eines Programms widerspiegeln. Für uns sind zunächst die von Python unterstützen Elemente interessant.

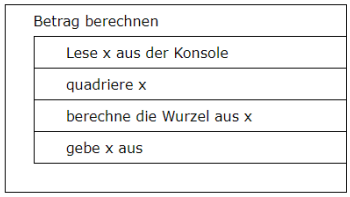
Figure 1: Weeks of coding...

3.1 Symbolik

im Folgenden wird de Symbolik von Struktogrammen kurz anhand von Minimalbeispielen erklärt. Diese Beispiele enthalten immer ein kleines Struktogramm mit dem passenden Codesnippet.

3.1.1 Anweisung

Eine Anweisung wird in einem rechteckigen Kasten geschrieben. Alle die Blöcke werden von oben nach unten abgearbeitet. Im folgenden Beispiel wird der Betrag einer Zahl berechnet und ausgegeben, welche zuvor eingelesen wurde.

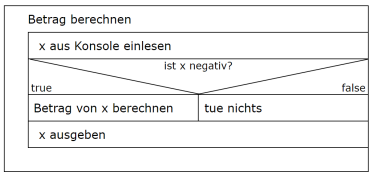



```
from math import sqrt

x = int(input("x eingeben: "))
x = x**2
x = sqrt(x)
print(x)
```

3.1.2 If-Abfrage

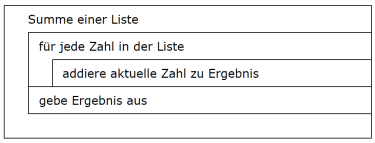
Eine If-Abfrage[1] stellt eine Verzweigung im Programm dar. Wenn eine Bedingung erfüllt ist, wird en bestimmter Teil Code ausgeführt.
Wir lesen eine Zahl ein. Falls die Zahl negativ ist, soll der Betrag berechnet werden. Abschließend wird die Zahl ausgegeben:




```
x = int(input("x eingeben: "))
if x < 0:
  x = abs(x)
print(x)
```

3.1.3 For-Schleife

For-Schleifen werden da verwendet, wo vorab bereits bekannt ist, wie viele Schleifendurchläufe gebraucht werden. Beispiele hierfür ist das iterieren über eine Liste oder das durchführen eines Vorgangs n-mal.
Wir nehmen an, dass in einer Liste eine Menge an Zahlen steht. Die For-Schleife soll die Zahlen aufsummieren. Abschließend soll die Summe ausgegeben werden:




```
num_lst = [1, 2, 3, 4, 5]
res = 0
for num in num_lst:
  res = res + num
print(res)
```

3.1.4 While-Schleife

While-Schleifen werden dort verwendet, wo die Abbruchbedingung der Schleife anfangs noch nicht direkt bekannt ist. Die Schleife wird so lange ausgeführt, wie die Bedingung im Schleifenkopf True ist.

[1] Achtung: Es wird IF-Abfrage genannt. If-Schleifen gibt es nicht!!!

3.2 Aufgaben

Die folgenden Aufgaben könnt ihr mit Hilfe des Tools *structorizer* (https://www.structorizer.com/struct.php) bearbeiten. Das Tool weist weitere Elemente für Struktogramme auf, die wir entweder nicht verwenden oder die nicht in Python vorhanden sind.

3.2.1 Warmup

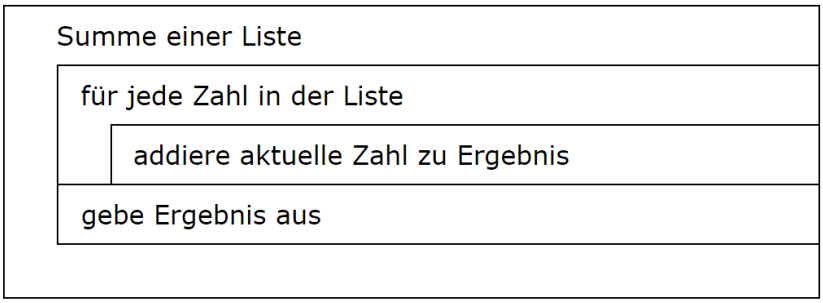
Um warm zu werden mit dem Tool, sollt ihr zunächst die Struktogramme aus 3.1 nachbauen.

3.2.2 Code zu Struktogramm

Erstellt aus eurem Code für die Aufgabe 2 (Dinge mit Listen und Schleifen) ein Struktogramm.

3.2.3 Struktogramm zu Code

Erstellt zu folgendem Struktogramm das Programm:



3.2.4 Würfeln

Es soll ein Programm entworfen werden, welches einen Würfel so lange würfelt, bis eine Zahl 3 mal hintereinander gefallen ist. Die Anzahl der benötigten Würfe ist am Ende auszugeben. Der Würfel hat 6 Seiten.
Erstellt zunächst das Struktogramm und implementiert eure Lösung anschließend.

3.2.5 Zahlenspiel

Ihr sollt ein kleines Zahlenspiel planen und implementieren. Das Spiel wird über die Konsole gespielt. Der Spieler muss eine Zahl (ganze Zahl) erraten, die der Computer sich zuvor ausdenkt. Hierbei wird die vom Spieler geratene Zahl über die Konsole eingelesen. Der Computer gibt daraufhin den Tipp, ob die geratene Zahl kleiner oder größer ist als die zu erratende Zahl. Sollte der Spieler die Zahl richtig erraten, wird das Programm mit einer passenden Ausgabe beendet.

Erstellt zunächst das Struktogramm des Spiels. Anhand dieses Struktogramms sollt ihr anschließend das Programm implementieren.

4 Aufgabe 3: Streichhölzer ziehen

In dieser Aufgabe soll ein bekanntes Streichholzspiel realisiert werden. Von einer Anfangsmenge von Streichhölzern nehmen zwei Spieler abwechselnd eins, zwei oder drei Hölzchen weg. Wer das letzte Hölzchen nehmen muß, hat verloren. Das Programm soll so gestaltet sein, daß ein Spieler gegen den Computer spielt.

Der Spieler kann dabei bei jedem seiner Zügen nach Gutdünken wahlweise eins, zwei oder drei Hölzchen nehmen. Die Züge des Computers seien zunächst in einer ersten Version so realisiert, daß er zufällig eins, zwei oder drei Hölzchen nimmt. Lediglich wenn nur noch vier Hölzchen oder weniger übrig geblieben sind, spielt der Computer so, daß er gewinnt. Bei vier Hölzchen beispielsweise nimmt er drei, damit der Spieler auf dem letzten Hölzchen sitzen bleibt.

Zufallszahlen kann man in python mit der Funktion **randint** aus der Bibliothek **random** generieren. Mit den Anweisungen:

```
import random

krasse_zahl = random.randint(1,10)
```

wird der Variable krasse_zahl eine Zufallszahl zwischen 1 und 10 zugewiesen.

Damit bei jedem Programmdurchlauf immer eine neue Folge von Zufallszahlen berechnet wird, muss der Zufallsgenerator einmal am Anfang des Programms initialisiert werden. Dies geschieht mit der Funktion **random.seed** Mit könnte man somit beispielsweise Zufallszahlen zwischen 1 und 3 erzeugen. Achte darauf, daß auch der Spieler immer nur eins, zwei oder drei Hölzchen nehmen darf, nicht mehr und nicht weniger.

Wenn das Programm läuft, überlegt euch eine optimale Strategie für den Computer. Der Computer soll also in einer verbesserten Version des Programms von Beginn an stehts den optimalen Zug ausführen. Er soll also immer so ziehen, daß er, wenn er die Chance zu gewinnen hat, diese auch nutzt. Die Darstellung auf der Console könnte dann z.B. wie folgt aussehen:

```
Die aktuelle Anzahl der Hoelzchen ist: 13
* * * * * * * * * * * * *
I I I I I I I I I I I I I
I I I I I I I I I I I I I
I I I I I I I I I I I I I
* * * * * * * * * * * * *
Gönn dir Hoelzchen, Brudi! (1, 2, 3):
```