Python cheat sheet 2

1 Turtle

Um Python's Fähigkeiten in eine spezifische Richtung zu erweitern, benutzt man die sogenannte Bibliotheken (manchmal auch Paketen genannt), das sind Sammlungen von nützlichen Befehlen für ein spezifisches Zweck. Es gibt mehr als 200,000 Python-Bibliotheken: für Mathematik, Zeichnen, Entwicklung von Computerspielen, Webseiten ... Wir werden die Bibliothek für einfaches Zeichnen `turtle` benutzen. Um diese Bibliothek zu benutzen, muss man am Anfang des Programms schreiben:

```
import turtle
```

So weiß Python, das wir über die übliche Befehle hinaus noch diejenige aus der Bibliothek `turtle` benutzen wollen. Um die neue Befehle zu benutzen, schreiben wir immer `turtle`, dann stellen wir ein Punkt und schreiben nachdem den Befehl, z.B. `turtle.forward(100)`. Einige nützliche Befehle sind hier gesammelt:

`.forward()`	*nach vorne für eine bestimmte Länge, z. B.* `turtle.forward(100)`
`.backward()`	*rückwärts für eine bestimmte Länge, z. B.* `turtle.backward(70)`
`.right()`	*rechts abbiegen für ein bestimmtes Winkel, z. B.* `turtle.right(90)`
`.left()`	*links abbiegen für ein bestimmtes Winkel, z. B.* `turtle.left(30)`
`.circle()`	*ein Kreis (oder ein Teil davon) bestimmter Große, z. B.* `turtle.circle(50, extent=180)`
`.goto()`	*spezifische Koordinaten besuchen, z. B.* `turtle.goto(20, 30)`
`.penup()`	*bis zum Befehl* `.pendown()` *nicht zeichnen*
`.pendown()`	*wieder anfangen zu zeichnen*
`.speed()`	*Geschwindigkeit zwischen 1 und 10 festlegen z. B.* `turtle.speed(10)`
`.color()`	*die Farbe auswählen, z. B.* `turtle.color("Green")`
`.bgcolor()`	*die Farbe des Hintergrunds auswählen z. B.* `turtle.bgcolor("Black")`
`.begin_fill()`	*die Zeichnung ab hier soll mit der Farbe gefüllt werden*
`.end_fill()`	*nun alles ab* `.begin_fill()` *mit der Farbe füllen*
`.hideturtle()`	*turtle nicht mehr anzeigen*
`.clearscreen()`	*alles bisher gezeichnetes löschen*
`.done()`	*nur einmal, am Ende benutzen, damit das Bild offen bleibt*

2 Erinnerung - for-Schleifen

Mit der `turtle` ist es einfach, ein Quadrat zu zeichnen. Das geht zum Beispiel so:

```
turtle.color("Blue")
turtle.forward(100)
turtle.left(90)
turtle.forward(100)
turtle.left(90)
turtle.forward(100)
turtle.left(90)
turtle.forward(100)
turtle.left(90)
```

Es geht aber auch eleganter. Offenbar wiederholen wir genau das Gleiche vier Mal. Das riecht nach einer `for`-Schleife. Nämlich dieser:

```
turtle.color("Blue")
for i in range(4):
    turtle.forward(100)
    turtle.left(90)
```

Der Vorteil? Man kann der zweite Code schneller verarbeiten - eine andere Länge oder ein anders Winkel ausprobieren, vielleicht sogar ein Sechseck statt Viereck zeichnen (wie?) - und muss dabei nur eine Stelle korrigieren.

3 Funktionen

Wir können den Quadrat aus dem letzten Absatz noch ein bisschen verbessern. Vielleicht wollen wir viele Quadrate zeichnen - einige größere, einige kleinere, in verschiedenen Farben ... immer den gleichen Code kopieren und leicht verarbeiten? Nicht so schön. Stattdessen definieren wir eine Funktion, die Quadrate malt.

```
def quadrat(farbe, lange):
    turtle.color(farbe)
    for i in range(4):
        turtle.forward(lange)
        turtle.left(90)
```

Diese Funktion können wir danach mit verschiedenen Farben und Größen anrufen. So kann ich jetzt in nur drei Zeilen drei Quadrate malen. Noch ein Vorteil: es ist ziemlich offenbar, was der Befehl `quadrat("Green", 30)` macht. Wenn man hingegen nur die for-Schleife sieht, muss man erst nachdenken, was das Ergebnis wird.

```
quadrat("Green", 30)
quadrat("Red", 50)
quadrat("Blue", 100)
```

Wie sieht eine Funktion aus? Sie beginnt immer mit dem Wort `def`, was bedeutet, dass wir eine neue Funktion definieren wollen. Danach kommt der Name der Funktion, in diesem Fall habe ich sie `quadrat` genannt. Es folgen die Klammern. In Klammern können ein oder mehrere Argumente stehen, in diesem Fall bloß `farbe` und `lange`. Das sind Variablen, die wir in der Funktion benutzen können - welches Wert sie haben werden, stellen wir aber erst fest, wenn wir die Funktion mit spezifischen Argumenten anrufen, z. B. `quadrat("Green", 30)`, was in diesem Fall bedeutet, dass `farbe = "Green"` und `lange = 30`. Manche Funktionen haben auch keine Argumente - in diesem Fall stellen wir nach dem Funktionsnamen einfach leere Klammern. Am Ende der Zeile kommt immer ein Doppelpunkt und die nächsten Zeilen - alle, die zur Funktion gehören - sind nach rechts verschoben.

4 Input

Manchmal will der Benutzer mit dem Programm interagieren bzw. wir brauchen eine Information von ihm. Er wird, z.B. gefragt, was sein Name ist. Was auch immer der Benutzer schreibt, das Programm kann das in eine Variable, z. B. `benutzer_name` speichern, und kann danach personalisierte Nachrichten drucken. Wenn wir den Benutzer bitten, eine Zahl anzugeben, müssen wir diese noch mit `int()` ins ein `integer`-Typ umwandeln, bevor wir sie in Rechenoperationen benutzen.

```
benutzer_name = input("Guten Morgen! Wie heißt du? ")
schwestern = int(input("Wie wiele Schwestern hast du? "))
brueder = int(input("Wie wiele Brüder hast du? "))
print(f"{benutzer_name} hat insgesamt {schwestern+brueder} Geschwister.")
```

5 If-Sätze

Ein if-else Satz bedeutet eine Spaltung im Programm. Etwas kann passieren - oder etwas anderes. Je nachdem, ob die Bedingung, die zwischen `if` und Doppelpunkt stehlt, stimmt. Falls ja, passirt, was under `if` steht, und alles unter `else` wird übersprungen. Falls nein, genau umgekehrt. Falls da noch `elif` Sätze stehen, wird erst überprüft, ob if stimmt, dann die elif Bedingungen und falls nichts stimmt, aktiviert sich wieder der else-Satz. Unten haben wir eine einfache Empfehlungsmaschine. Der Benutzer informiert sie über die aktuelle Temperatur und je nach dem Wetter empfehlt sie ein Getränk oder Essen. Natürlich könnte man noch mehr Fragen stellen und mehr Optionen anbieten.

```
temperatur = int(input("Wie viel Grad ist es heute? "))
if temperatur > 25:
    print("Iss unbedingt ein Eis.")
elif temperatur < 5:
    print("Bestelle eine heiße Schokolade.")
else:
    print("Trink am besten frischen Orangensaft.")
```