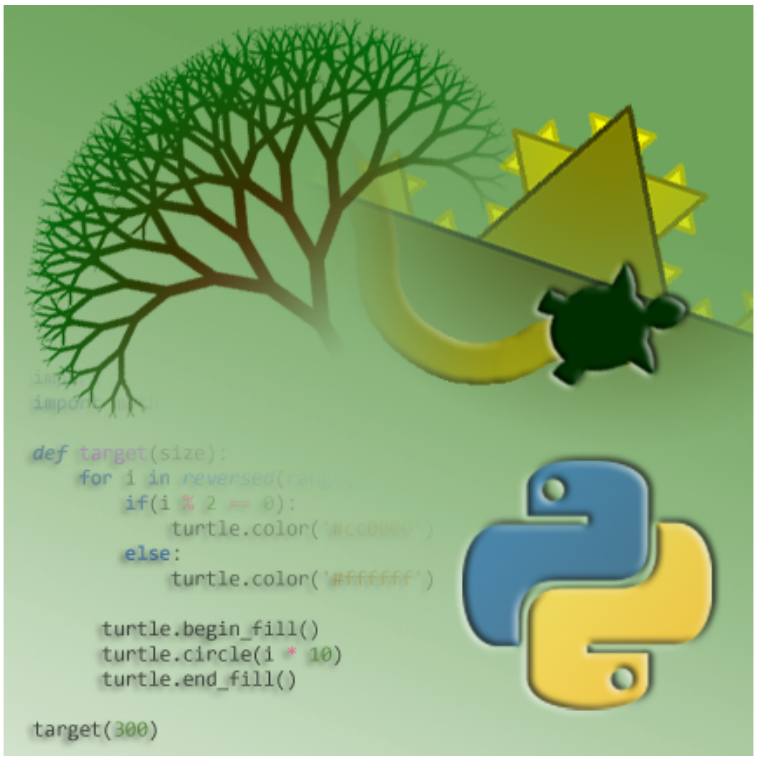
dEIn Labor.

Projekt:
Python Programmierung

Zeichnen mit Schildkröten



das Elektrotechnik- und Informatik-Labor der Fakultät IV
http://www.dein-labor.tu-berlin.de

berlin

Handout zum Projekt:

Python Programmierung

Zeichnen mit Schildkröten

Inhaltsverzeichnis

Python gehört zu den populärsten und weltweit meist verwendeten Programmiersprachen. Dieser Workshop bietet Euch einen Einstieg in die textbasierte Programmierung. Dazu wird das Turtle-Modul von Python verwendet, um Grundlegende Konzepte, wie Variablen, Schleifen und Verzweigungen zu erlernen.

1. Hello World

Mein erstes Python-Programm

a) Öffne als Erstes einen beliebigen Texteditor (z.B. Notepad++) und lege eine neue (noch leere) Textdatei an.Speichere diese auf dem Desktop unter dem Namen `hello.py`

b) Tippe nun den folgenden Befehl:

```
print("Hello world")
```

und speichere die Datei ab.

c) Teste nun das Programm:
- Öffne eine Konsole: Start → Tippe „cmd“ → *Enter*
- Navigiere zum Desktop, indem du `cd Desktop` eintippst und mit *Enter* bestätigst[1].
- Um dein Programm zu starten, tippe: `python hello.py`
  In der Konsole sollte nun der Text „Hello World“ angezeigt werden.

d) Mit Programmiersprachen lassen sich auch mathematische Berechnungen durchführen. Füge folgenden Befehl unter das letzte `print()` ein:

```
print((1 + 4) * 2)
```

Speichere die Datei erneut ab und teste nochmal das Programm wie in Schritt c)

e) Häufig möchte man beim Programmieren zur Dokumentation Kommentare oder Beschreibungen zu bestimmten Codezeilen in den Programmcode schreiben. Dieser Text ist aber nur für menschliche Nutzer gedacht und soll vom Computer ignoriert werden. Solche Kommentare leitet man mit dem Zeichen `#` ein. Alles was in einer Zeile dahinter steht, wird vom Computer nicht beachtet.

Beispiel:

```
# Kommentar: All dieser Text wird vom Computer ignoriert
print((1 + 4) * 2) # Ein Kommentar muss nicht am Zeilenanfang stehen
```

Dies kann auch dazu genutzt werden, einzelne Codezeilen *auszukommentieren*, damit sie beim Testen vorübergehend ignoriert werden. Dies ist später sehr nützlich!

[1] Falls das Programm nicht auf dem Desktop liegt, musst du erst in den entsprechenden Ordner navigieren. Wie das geht, wird hier beschrieben:
http://de.wikihow.com/Ordner-wechseln-in-der-Eingabeaufforderung

2. Zeichnen mit Turtle

Erste Schritte

Turtle ist ein Modul in Python, das sich verhält wir ein Zeichenbrett. Es besitzt Funktionen wie `turtle.forward(...)` und `turtle.left(...)`, welche die Schildkröte umher bewegen. Dabei hinterlässt sie eine Linie.

f) Als Erstes müssen wir die Programmdatei erzeugen:
- Öffne einen beliebigen Texteditor (z.B. Notepad++) und lege eine neue (noch leere) Textdatei an[2].
- Speichere diese dann auf der Festplatte und verwende die Dateieindung *.py, also etwa: *meinErstesProgramm.py*

g) Damit das Turtle-Modul benutzt werden kann, muss es importiert werden.
Tippe dazu als erste Zeile Folgendes:

```
import turtle
```

h) Nun kann das Modul verwendet werden.
Die folgenden 2 Befehle veranlassen, dass sich die Schildkröte um 250 Pixel vorwärts bewegt und dann um 60° nach links dreht:

```
turtle.forward(250)
turtle.left(60)
```

i) Teste nun das Programm, indem du auf die Quellcodedatei doppelklickst.

j) Nachdem die Befehle ausgeführt wurden, schließt sich das Programm direkt.
Um dies zu verhindern, schreibe den Befehl:

```
turtle.exitonclick()
```

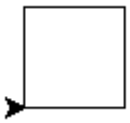
ganz ans Ende der Datei.

k) Die Schildkröte wird aktuell als einfacher Pfeil dargestellt. Mit dem Befehl `turtle.shape("turtle")` kann jedoch eine richtige Schildkröte daraus gemacht werden.
Das gesamte Programm sieht nun in etwa so aus:

```
import turtle
turtle.shape("turtle")
turtle.forward(250)
turtle.left(60)
turtle.exitonclick()
```

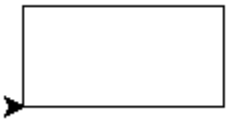
[2] Textverarbeitungsprogramme wie Word sind sind ungeeignet, da sie nicht reinen Quelltext, sondern Text mit Formatierung (Größe, Farbe etc...) speichern, was wir für die Programmierung nicht brauchen.

Übung a) Zeichne ein Quadrat so wie im folgenden Bild:



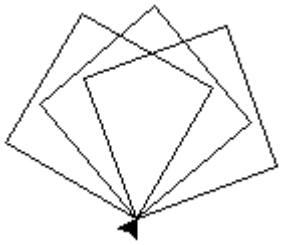
Dafür brauchst du vermutlich einen rechten Winkel, welcher 90˚ beträgt.

Übung b) Zeichne ein Rechteck, ähnlich zu diesem:



Bonus: Zeichne ein Dreieck! In einem gleichseitigen Dreieck (alle Seiten haben die gleiche Länge) hat jede Ecke einen Winkel von 60 Grad.

Übung c) Zeichne 3 schräg zur Seite gekippte Quadrate. Experimentiere mit den Winkeln zwischen den einzelnen Quadraten.



Das Bild zeigt 3 Drehungen um jeweils 20 Grad. Probiere auch einmal andere Werte aus!

Übung d) Die geometrische Form kann mit den Funktionen `turtle.width(...)` und `turtle.color(...)` verändert werden.

Schreibst du z.B.:

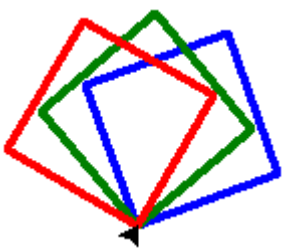
```
turtle.width(5)
```

werden ab jetzt alle Linien mit der Breite von 5 Pixeln gezeichnet.

Wenn du den Befehl

```
turtle.color("green")
```

verwendest, sind ab jetzt alle Linien grün.



Versuche, eine Grafik ähnlich zu dieser zu erzeugen!

3. Variablen

Wenn du mit den Winkeln und Seitenlängen experimentierst, musst du jedes Mal an verschiedenen Stellen den Code verändern, das ist unpraktisch!

An dieser Stelle kommen Variablen ins Spiel: Du kannst ab nun Python befehlen, jedes Mal wenn du eine bestimmte Variable verwendest, an dieser Stelle etwas anderes einzusetzen. Dieses Konzept ist ähnlich zur Algebra, wo du schreiben könntest: x sei 5. Dann ist x * 2 natürlich 10.

In der Syntax von Python hat das die gleiche Bedeutung:

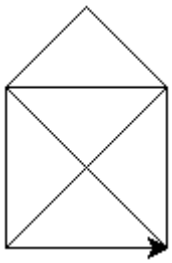
```
x = 5
```

Wenn du nach diesem Befehl `print(x)` ausführst, wird der Wert von x ausgegeben: 5. Du kannst das Gleiche auch mit *turtle* kombinieren:

```
turtle.forward(x)
```

Übung a) Sieh dir noch einmal das Programm mit den gekippten Quaddraten aus Übung 2c) an. Lege zwei Variablen an und nenne sie `angle` (für den Winkel) und `length` (für die Seitenlänge). Ersetze nun die Zahlen in den Aufrufen `forward()` und `left()` durch die zwei Variablen.

Übung b) Zeichne das Haus des Nikolaus:



Es sollen zwei Variablen verwendet werden:
Eine für die Länge der Rechteckseiten und eine für die Länge der Diagonalen, die von der Länge der Rechteckseiten abhängig ist (Satz des Pythagoras).

Um die Quadratwurzel einer Zahl zu berechnen, muss das *math* Modul importiert und die Funktion `math.sqrt()` aufgerufen werden. Das Quadrat einer Zahl wird mit dem Operator `**2` berechnet.

Als Hilfestellung ist der Anfang des Programms vorgegeben. Schreibe es fertig!

```
import turtle
import math

length = 100    # Seitenlänge des Quadrats
diag = math.sqrt(a**2 + b**2)  # Länge der (langen) Diagonalen

turtle.left(45)
turtle.forward(diag)
turtle.left(135)
turtle.forward(length)
turtle.left(135)
turtle.forward(diag)
turtle.left(135)
  ...
```

4. Schleifen

Wie du sicher bemerkt hast, enthalten unsere Programme oft Wiederholungen. Programmiersprachen bieten dafür ein mächtiges Konzept: Schleifen.

Probier einmal folgendes Beispiel aus. Was ist die Ausgabe?

```
for name in "John", "Sam", "Jill":
    print("Hello " + name)
```

Schleifen sind hilfreich, um mehrere Male etwas zu tun und den Programmcode dafür nur einmal zu schreiben, wie etwa beim Zeichnen der Umrandung einer geometrischen Form.

Hier ist eine andere Version einer Schleife:

```
for i in range(10):
    print(i)
```

Die `range(n)`-Funktion kann man als eine Abkürzung für `0, 1, 2, ..., n-1` sehen.

Du kannst auch Elemente deiner Wahl in einer Schleife verarbeiten:

```
total = 0
for i in 5, 7, 11, 13:
    print(i)
    total = total + i
print(total)
```

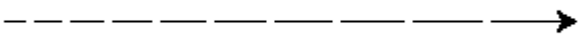
Schreib das Beispiel ab und und überprüfe, ob es so funktioniert wie du dir vorstellst.

Wichtig: Alle Zeilen, die wiederholt werden sollen, müssen eingerückt werden. Dadurch weiß Python, welche Zeilen zur for-Schleife gehören und welche danach kommen.

Übung a) Zeichne eine gestrichelte Linie, mit Hilfe einer Schleife:

Mit `turtle.penup()` erreichst du, dass die Schildkröte beim Bewegen nicht zeichnet, mit `turtle.pendown()` hinterlässt sie bei folgenden Bewegungen wieder eine Linie.

Übung b) Lasse die Striche immer größer werden:



Übung c) Die Quadrate aus 2.) erforderten viele wiederholte Textzeilen. Schreibe ein Programm, das ein Quadrat wie in Übung 2a), aber mit weniger Codezeilen, zeichnet!

Übung d) Verschachtele 2 Schleifen, um die Quadrate aus Übung 2c) zu zeichnen.

5. Funktionen

Mit Funktionen hat man die Möglichkeit, mehrere Anweisungen in einer einzigen Anweisung zusammenzufassen.

In Python kann eine Funktion mit dem Schlüsselwort `def` definiert werden:

```
def line_without_moving():
    turtle.forward(50)
    turtle.backward(50)
```

Diese von uns definierte Funktion heißt `line_without_moving` und ist praktisch eine Abkürzung für zwei Bewegungen: ein Schritt vorwärts und ein Schritt rückwärts.

Um sie zu benutzen *(aufzurufen)*, schreib ihren Namen gefolgt von runden Klammern:

```
line_without_moving()
turtle.right(90)
line_without_moving()
turtle.right(90)
line_without_moving()
```

Funktionen können auch innerhalb von anderen Funktionen verwendet werden. Eine Funktion, die rechts abgebildete sternförmige Figur zeichnet, könnte beispielsweise so definiert und aufgerufen werden:

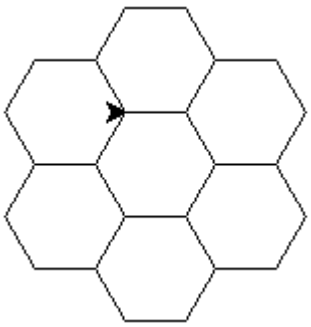
```
def star():
    for i in range(5):
        line_without_moving()
        turtle.right(360 / 5)

star()
```

Wichtig: Beachte auch hier die korrekte Einrückung, damit Python erkennt, welche Zeilen zur Funktion und welche zur `for`-Schleife innerhalb der Funktion gehören.

Übung a) Schreibe eine Funktion, die ein Quadrat zeichnet. Versuche anschließend, die Funktion zu verwenden, um das Programm mit den gekippten Quadraten aus Übung 2c) zu verbessern.

Übung b) Schreibe eine Funktion die ein Hexagon (Sechseck) zeichnet.

Übung c) Verwende diese dann, um eine Honigwabe zu zeichnen.



6. Funktionen mit Parametern

Unsere bisherigen Funktionen machen jedes Mal genau dasselbe: Die hexagon-Funktion aus Übung 5d) kann beispielsweise nur Sechsecke einer Größe zeichnen. Um dies zu ändern, benötigen wir Parameter. So können die Werte innerhalb der Funktion bei jedem Aufruf anders sein.

Zum Beispiel können wir die Funktion `line_without_moving()` verbessern, indem wir ihr einen Parameter übergeben:

```
def line_without_moving(length):
    turtle.forward(length)
    turtle.backward(length)
```

Der Parameter fungiert als Variable, die nur innerhalb der Funktion bekannt ist. Wir verwenden die Funktion, indem wir sie mit dem Wert für den Parameter aufrufen:

```
line_without_moving(30)
turtle.right(90)
line_without_moving(40)
```

Dadurch zeichnet sie nicht jedes Mal eine Linie der Länge 50, sondern der Länge, die wir als Parameter angegeben haben (in obigem Beispiel also 30 und 40).

Wir haben von Anfang an bereits Funktionen mit Parametern verwendet, wie z.B. `turtle.forward()` oder `turtle.left()`.

Für eine Funktion können beliebig viele Parameter definiert werden. Die einzelnen Parameter sind dabei durch Kommata getrennt und haben alle unterschiedliche Namen:

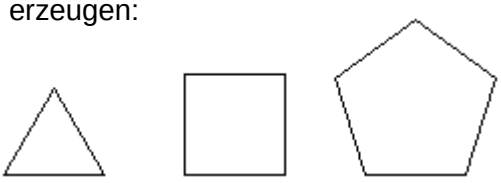
```
def tilted_line_without_moving(length, angle):
    turtle.left(angle)
    turtle.forward(length)
    turtle.backward(length)
```

Übung a) Schreibe eine Funktion, die ein Sechseck beliebiger Größe zeichnet.

Übung b) Schreibe eine Funktion, die eine Form mit beliebiger Größe und beliebiger Anzahl von Seiten zeichnet:

```
def shape(length, edges):
    ...
```

Aufrufe von `shape(50,3)`, `shape(50,4)`, `shape(50,5)` sollten beispielsweise folgende Formen erzeugen:



Tipp: Die Summe der Außenwinkel in jeder Form ist immer 360 Grad!

7. Verzweigungen

Verzweigungen bieten die Möglichkeit, bestimme Programmabschnitte nur dann auszuführen, wenn eine Bedingung gilt.

Konzeptionell wird eine Verzweigung folgendermaßen formuliert:

```
if Bedingung:
    # Code, der ausgeführt wird, wenn die Bedingung wahr ist
else:
    # Code, der ausgeführt wird, wenn die Bedingung falsch ist
```

Die *Bedingung* ist dabei ein Ausdruck, der zu `True` oder `False`, also *wahr* oder *falsch* ausgewertet wird. Zum Beispiel kann man testen, ob eine Variable einen bestimmten Wert hat (ist gleich: `==`), oder ob eine Zahl größer als (`>`) oder kleiner als gleich (`<=`) einer anderen Zahl ist.

Probiere einemal das folgende Beispiels aus und sieh dir an, wie sich das Programm verhält, wenn du für `zahl` Werte größer oder kleiner als `50` wählst:

```
zahl = 75
if zahl > 50:
    print("Die Zahl ist größer als 50")
else:
    print("Die Zahl ist kleiner gleich 50")
```

Der `else`-Block ist bei einer if-Abfrage optional. Falls du ihn auslässt und die Bedingung als `False` ausgewertet wird, passiert einfach nichts.

Mit der `input()`-Funktion kann der Benutzer aufgefordert werden, einen Text einzutippen. Dies können wir verwenden, um die Schildkröte zu steuern. Um es nicht zu kompliziert zu machen, erlauben wir nur die zwei Befehle „links“ und „rechts“.

Schreibe folgende Funktion ab und rufe sie mindestens einmal mit `move()` auf:

```
def move():
    direction = input("Links oder rechts?")
    if direction == "links":
        turtle.left(60)
    if direction == "rechts":
        turtle.right(60)

    turtle.forward(50)
```

Bei jedem Aufruf von `move()` wirst du aufgefordert, entweder „links“ oder „rechts“ einzutippen. Was passiert, wenn du etwas ganz anderes eintippst?

Übung a) Benutze if-Abfragen, um in dem Beispiel mit den farbigen gekippten Quadraten aus Übung 2d) die Farbwahl zu steuern.