Themen

Selbst definierte Funktionen

Häufig müssen in Programmen Abfolgen von Anweisungen mehr als einmal ausgeführt werden. Um diese Programmabschnitte nicht wiederholen zu müssen, verwendet man Funktionen.

Funktionen werden im oberen Teil eines Programms definiert. Im Hauptprogramm muss dann nur noch der Funktionsname aufgerufen werden, damit die Anweisungen ausgeführt werden.

Längere Programme erhalten dadurch eine übersichtliche Struktur. Sprechende Funktionsnamen können zudem helfen, die Wirkungsweise eines Programms zu verstehen.

Definition mit def

Die Definition einer Funktion begint mit der Anweisung `def` und dem Namen der Funktion. Der Name einer Funktion kann frei gewählt werden. Nach dem Namen folgen runde Klammern und ein Doppelpunkt. Die Anweisungen, die zur Funktion gehören, werden eingerückt.

In der Klammer können ein oder mehrere Parameter aufgelistet sein, die innerhalb der Funktion benötigt werden.

```
#Definition der Funktion quadrat
def quadrat(seite):
    ergebnis = seite * seite
    return ergebnis

#Hauptprogramm
seite = input("Seitenlänge")

flaeche = quadrat(seite)
print "Fläche:",flaeche
```

Im Hauptprogramm wird die Funktion aufgerufen. In unserem Beispiel wird das Ergebnis der Funktion der Variablen **flaeche** zugewiesen und ausgegeben.

return

Damit das innerhalb der Funktion berechnete Ergebnis im Hauptprogramm zugewiesen und ausgedruckt werden kann, wird die Anweisung `return` benötigt.

`return` erzeugt einen Rückgabewert, der als Ergebnis der Funktion an das Hauptprogramm zurückgeliefert wird. Die Variable `ergebnis` wird dabei nicht mit übergeben.

Zugleich beendet die Anweisung `return` die Ausführung der Funktion.

Lokale und globale Variablen

Jede Funktion bildet für die Variablen einen geschlossenen, lokalen Namensraum. Das bedeutet, dass man auf Variablen, die innerhalb einer Funktion definiert wurden, aus anderen Funktionen oder dem Hauptprogramm heraus nicht zugreifen kann.

In unserem ersten Beispiel trifft das auf die Variable `ergebnis` zu. Sie wird in der Funktion quadrat definiert. Im Hauptprogramm wird aber eine neue Variable benötigt, um die berechnete Fläche auszugeben.

Möchte man eine Variable sowohl innerhalb einer Funktion als auch im restlichen Programm verwenden, muss sie in der Funktion den Status `global` erhalten.

Mehrere Variablen, die als `global` deklariert und mit `return` übergeben werden sollen, werden – durch Komma getrennt – aneinandergereiht.

```
#Definition der Funktion rechteck
def rechteck(seite_a,seite_b):
    #Status global zuweisen
    global flaeche, umfang
    flaeche = seite_a * seite_b
    umfang = 2 * seite_a + 2 * seite_b
    return flaeche, umfang

#Hauptprogramm
seite_a = input("Seitenlänge a")
seite_b = input("Seitenlänge b")

rechteck(seite_a,seite_b)
#Variable aus Funktion
print "Fläche:",flaeche
print "Umfang:",umfang
```

Globale Variablen können nach Ablauf der Funktion auch im Hauptprogramm verwendet und beispielsweise ausgegeben werden.

Selbst definierte Funktionen

Aufgabe 1

Nimm in diesem Programm die folgenden kleinen Veränderungen vor. Beschreibe, was daraufhin beim Ausführen des Programms geschieht, und erkläre, warum das so ist.

```
def quadrat(seite):
    ergebnis = seite * seite
    return ergebnis

seite = input("Seitenlänge")

flaeche = quadrat(seite)
print flaeche
```

a) Kommentiere die Zeile `return ergebnis` aus.

b) Ändere die letzten beiden Zeilen des Hauptprogramms wie folgt:

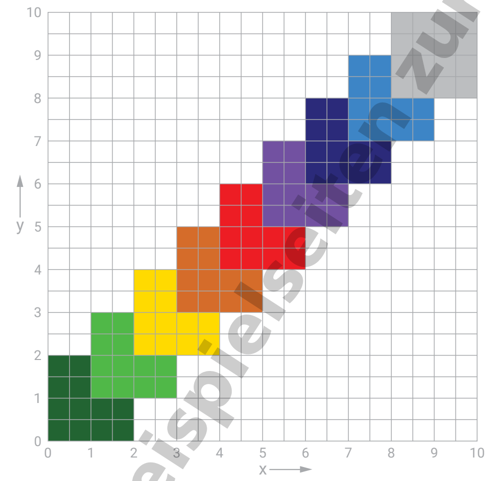
```
quadrat(seite)
print ergebnis
```

c) Fasse die letzten beiden Zeilen des Hauptprogramms zusammen zu

```
print quadrat(seite)
```

Aufgabe 2

Erstelle ein Programm, das neun in einer zufälligen RGB-Farbe gefärbte Quadrate auf diese Weise diagonal anordnet.
(Das Kästchenraster dient nur der Orientierung.)



a) Importiere dazu die benötigten Bibliotheken, setze den Anfangswert für den Zufallsgenerator und lass ein Grafikpanel mit den Parametern wie im Bild zeichnen.

b) Definiere eine Funktion, die ein buntes Quadrat zeichnet. Die RGB-Farbe soll aus zufälligen Werten für Rot, Grün und Blau zusammengesetzt werden.

c) Im Hauptprogramm soll zunächst der untere linke Eckpunkt des ersten Quadrats auf 0, 0 gesetzt werden.

d) In einer Schleife sollen dann nacheinander die neun Quadrate gezeichnet werden.
Die oberen rechten Eckpunkte der Quadrate sollen jeweils ausgehend vom aktuellen linken unteren Eckpunkt definiert werden.
Nach dem Ausführen der Funktion sollen die Koordinaten des linken unteren Eckpunktes um 1 nach oben und nach rechts verändert werden.

Aufgabe 3

Erstelle ein Programm, das eingegebene Wörter auf darin enthaltene, zufällig ausgewählte Buchstaben prüft.

a) Importiere dazu die benötigte Bibliothek und setze den Anfangswert für den Zufallsgenerator.

b) Definiere eine Funktion, die einen zufälligen Buchstaben von a bis z auswählt und als Rückgabewert an das Hauptprogramm übergibt.

c) Im Hauptprogramm soll mit Hilfe der Funktion drei Variablen nacheinander jeweils ein ausgewählter Buchstabe zugeordnet werden.

d) Der Nutzer soll zur Eingabe eines Wortes aufgefordert werden. In der Aufforderung sollen die drei Buchstaben enthalten sein, beispielsweise „Wort mit abc eingeben“.

e) Es soll geprüft werden, ob das eingegebene Wort alle drei Buchstaben enthält. Falls das so ist und auch falls das nicht so ist, sollen entsprechende Meldungen ausgegeben werden.

Programmierfehler (Bugs)

Beim Programmieren passieren sehr leicht Fehler. Selbst Profis gelingt es selten, auf Anhieb einen vollkommen fehlerfreien Programmcode zu schreiben. Programmierfehler – Profis nennen sie Bug (von englisch bug „Wanze, Ungeziefer“) – führen zu Fehlfunktionen, Programmabstürzen oder Sicherheitslücken. Das Suchen und Beseitigen von Fehlern ist daher ein wichtiger Schritt in der Softwareentwicklung.

Bei der Ausführung eines Programms innerhalb der Entwicklungsumgebung, wird es durch den so genannten Compiler (von englisch compile „zusammentragen“) in eine Form übersetzt, die vom Computer verstanden und ausgeführt werden kann. Viele häufig vorkommende Fehler werden dabei vom Compiler bereits erkannt und führen zu Fehlermeldungen der Entwicklungsumgebung. Diese Fehler werden daher auch als Compilerfehler bezeichnet. Damit man in den eigenen Programmen Compilerfehler besser findet, hilft es, die wichtigsten Arten von Programmierfehlern zu kennen.

Lexikalische Fehler

Werden in einem Programmcode Zeichen verwendet, die nicht zum verwendeten Alphabet gehören, spricht man von einem lexikalischen Fehler. In unserem Beispiel wird das griechische π als Zeichen erkannt, das nicht zum erlaubten Zeichensatz gehört.

```
radius = 10
flaeche = π * radius ** 2
print flaeche
```

```
Ungültiges Zeichen: 'π'/'[960]' [line 2]
```

Syntaktische Fehler (Syntaxfehler)

Jede Programmiersprache hat ihre eigenen Regeln für die richtige Schreibweise der Anweisungen – die Syntax. Sie ist mit den Regeln für Rechtschreibung und Grammatik vergleichbar. Fehlende Doppelpunkte, Einrückungen, Anführungszeichen oder Klammern sind die häufigsten syntaktischen Fehler. In unserem Beispiel ist es ein Operator, der für Zuweisungen nicht zulässig ist.

```
radius = 10
flaeche == 3.14 * radius ** 2
print flaeche
```

```
Verwende ein einzelnes Gleichheits-
zeichen '=' für Zuweisungen. [line 2]
```

Semantische Fehler

Die Semantik wird auch als Bedeutungslehre bezeichnet. In einem semantisch korrekten Programmcode sind alle Anweisungen so formuliert, wie es ihrer Bedeutung und der Programmlogik entspricht. Häufige semantische Fehler sind fehlende Variablendefinitionen, eine falsche Reihenfolge der Parameter, das Aufrufen einer Funktion vor deren Definition oder ein fehlendes `return` bei einer Funktion. In unserem Beispiel ist die Variable `pi` nicht definiert.

```
radius = 10
flaeche = pi * radius ** 2
print flaeche
```

```
Der Name 'pi' ist nicht definiert
oder falsch geschrieben.
```

Logische Fehler

Programme mit logischen Fehlern laufen ohne Fehlermeldungen durch, liefern aber falsche Ergebnisse. Entsprechend sind diese Fehler am schwersten zu finden. Häufige logische Fehler sind vertauschte Rechenoperationen oder fehlerhafte Berechnungsformeln wie in unserem Beispiel. Auch Denkfehler beim Aufbau von Programmen zählen zu den logischen Fehlern.

```
radius = 10
flaeche = 3.14 * radius * 2
print flaeche
```

```
62.8
```

Die Grenzen zwischen den Fehlerarten sind häufig etwas verschwommen. So ist das Komma in der Zahl `3.14` sowohl ein Syntaxfehler als auch ein semantischer Fehler, da er zu einer Fehlinterpretation der Zahl und damit zu einem völlig falschen Ergebnis führt.

```
radius = 10
flaeche = 3,14 * radius ** 2
print flaeche
```

```
(3, 1400)
```

Programmierfehler (Bugs)

Aufgabe 1

a) Markiere alle Fehler in diesem Programm.

b) Notiere jeweils die Fehlermeldung und die Art der Fehler.

```
from random import
from GPanel import *
seed(
makegpanel(0, 10, 0, 10)
def KreisZeichnen(r)
    rot = randint(0,255)
    grün = randint(0,255)
    blau = randint(0,255)
    setColor(rot,gruen,blau)
    fillCircle(R)

Hauptprogramm
x = 0
y == 0
for i in range1,5):
x = x + i
    y = y + i
    r = i
    pos x,y)
    KreisZeichnen(r)
```

Aufgabe 2

Im folgenden Programm sind sechs Zeilen markiert, deren Fehlen im weiteren Verlauf des Programms einen semantischen Fehler verursacht.

Beschreibe für jede dieser markierten Zeilen, was passiert, wenn sie fehlt und wo sich dieser Fehler auswirkt.

```
from gpanel import *  ①
makeGPanel(-10, 10, -6, 14)  ②
def BogenZeichnen(radius):  ③
    fillArc(radius,0,180)
farben = ["Red","DarkOrange","Gold",
"LimeGreen","RoyalBlue","BlueViolet",
"DeepPink","WhiteSmoke"]  ④
for i in range(0,8):
    farbe = farben[i]  ⑤
    setColor(farbe)
    radius = 9 - i  ⑥
    BogenZeichnen(radius)
```

Aufgabe 3

Das folgende Programm soll dieses Gesicht zeichnen.

a) Markiere die logischen Fehler im Programm, die das gewünschte Ergebnis verhindern.

b) Korrigiere das Programm.

c) Übertrage deine Korrekturen in das Programm Aufgabe_13-3_Gesicht.py. Liefert das Programm nun das gewünschte Ergebnis?

```
from gpanel import *
makeGPanel(-10, 10, -10, 10)

rot = 10
gruen = 150
blau = 200
setColor(blau, gruen, rot)
fillCircle(5)

pos(-2.5, 2)
pos(2.5, 2)
fillCircle(1)
fillCircle(1)
setColor("white")
fillTriangle(-0.75,-0.5,0.75,-0.5,0,1)
pos(-1.5, 0)
fillArc(2.5, 0, 180)
```

Aufgabe 4

Das Programm Aufgabe_13-4_Kueken.py enthält in fast jeder Zeile einen Syntaxfehler.

Finde und korrigiere die Fehler.

Führe das Programm testweise aus. Zeichnet es das Küken?