4. Funktionen und Module

In den Kapiteln zuvor haben wir Funktionen und Module kennen gelernt. Nun lernen wir, wie eigene Funktionen und Module erstellt werden.

Eigene Funktionen

Warum werden eigene Funktionen erstellt? Die Gründe sind sehr verschieden, warum es Sinn macht Funktionen zu definieren:

a.) Mehrere Prozesse lassen sich zu einem Namen zusammenfassen
b.) Ein längeres Programm wird dadurch kürzer
c.) Die Funktionsnamen können auf die Prozesse hindeuten
d.) Es wird vermieden den Code doppelt zu schreiben

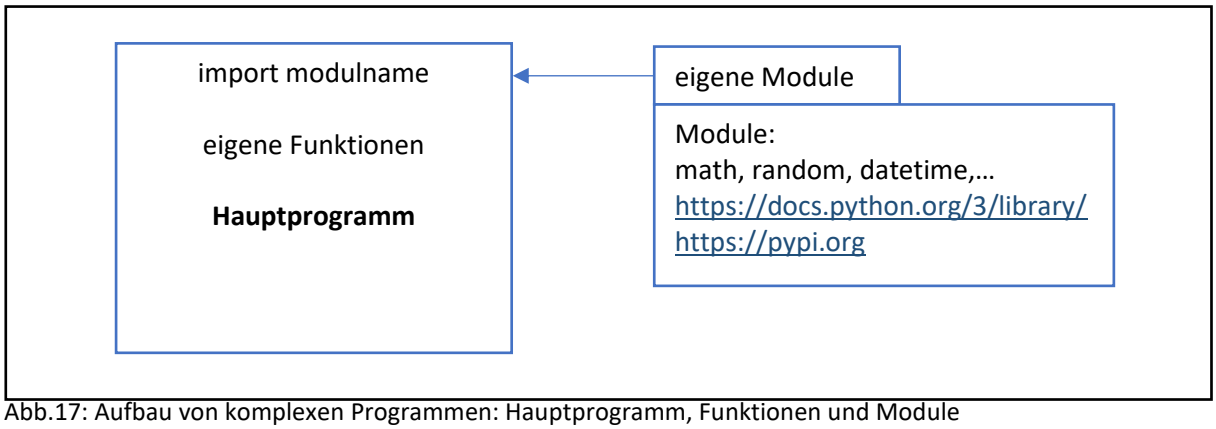

Abb.17: Aufbau von komplexen Programmen: Hauptprogramm, Funktionen und Module

Man unterscheidet zwei Arten von Funktionen

a.) Funktion ohne Rückgabewert
b.) Funktion mit Rückgabewert

Mit dem Schlüsselwort **def** wird eine neue Funktion eingeleitet. Dem **Funktionsnamen** folgen die runden Klammern, gefolgt von einem **Doppelpunkt**. In der Klammer werden die **Argumente** mit Kommas getrennt aufgelistet. Die darauffolgenden Zeilen müssen eingerückt sein, ansonsten gehören sie nicht mehr zur Funktion.

def funktionsname(Argumente):
    Anweisung

Die neue Funktion wird oberhalb des Programms definiert, so dass diese im Hauptprogramm genutzt werden kann. Die Funktionen können ausgelagert werden (siehe **eigene Module**).

Schauen wir uns nun die verschiedenen Arten von Funktionen (ohne/mit Rückgabewert) anhand nachfolgender Beispiele an:

Funktion ohne Rückgabewert

```
1 def ausgabe():
2     print("Hallo Welt!")
3 ausgabe()
```

Ausgabe: Hallo Welt!

Funktion ohne Rückgabewert mit einem Parameter

```
1 def mult2(eingabe):
2     eingabe=2*eingabe
3     print("Verdopple ich die Zahl, erhalte ich ", eingabe)
4 zahl = int(input("Gib eine ganze Zahl ein: "))
5 mult2(zahl)
```

Ausgabe: Gib eine ganze Zahl ein: 5
Verdopple ich die Zahl, erhalte ich 10

Die **Argumente** der Funktion können mit benannten Parametern und Vorgabewerten eingesetzt werden. Die definierte Reihenfolge der Argumente muss beim Aufruf nicht eingehalten werden, falls die Parameter mit Ihrem Namen übergeben werden.

Funktion ohne Rückgabewert mit benannten Parametern

```
1 def folge(anfangswert, endwert, schrittweite=1):
2     for x in range(anfangswert, endwert, schrittweite):
3         print(x)
4     print("Die Folge ist beendet.")
5 folge(4,9)
```

Ausgabe: 4 5 6 7 8 Die Folge ist beendet.

Funktion mit Rückgabewert

Um die Ergebnisse für den weiteren Programmablauf nutzen zu können, muss die Funktion ein Ergebnis zurückgeben. Dies erfolgt mit der **return** Anweisung. Im Unterschied zu anderen Programmiersprachen können Funktionen in Python mehr als einen Rückgabewert liefern.

```
1 def viereck(a, b):
2     flaeche = a * b
3     umfang = 2 * a + 2 * b
4     return flaeche, umfang
5 f, u = viereck(2,5)
6 print("Die Fläche beträgt: ", f ,"qcm und der Umfang beträgt: ", u, "cm")
```

Ausgabe: Die Fläche beträgt: 10 qm und
der Umfang beträgt: 14 cm

Rekursive Funktionen

Eine rekursive Funktion ruft sich immer wieder selbst auf. Damit dies nicht zu einer endlosen Funktion führt, muss es eine Bedingung geben, die der Rekursion ein Ende setzt. Der erste Aufruf der rekursiven Funktion erfolgt mit einem Startwert aus dem Hauptprogramm heraus.

```
1  def halbieren(wert):
2      print(wert)
3      wert = wert / 2
4      if wert > 0.5:
5          halbieren(wert)
6  halbieren(5)
```

Ausgabe: 5, 2.5, 1.25, 0.625

Erstellen wir nun ein Python-Programm, womit eine ganze Zahl (>1) in eine Binärzahl mit der Methode „Division durch zwei mit Rest“ umgewandelt wird. Der Quotient der Division durch 2 wird als ganze Zahl notiert und immer wieder mit 2 geteilt, bis der Quotient 0 ist. Die Ausgabe soll jeweils pro Schritt den Quotienten mit dem Rest anzeigen.

Hinweis:
Der binäre Rest ergibt bei geraden Quotienten den Rest 0 und bei ungeraden Quotienten den Rest 1. Die Ziffernfolge der Restwerte von unten nach oben ergibt die Binärzahl. Zur Kontrolle kann die eingebaute Funktion **bin()** genutzt werden.

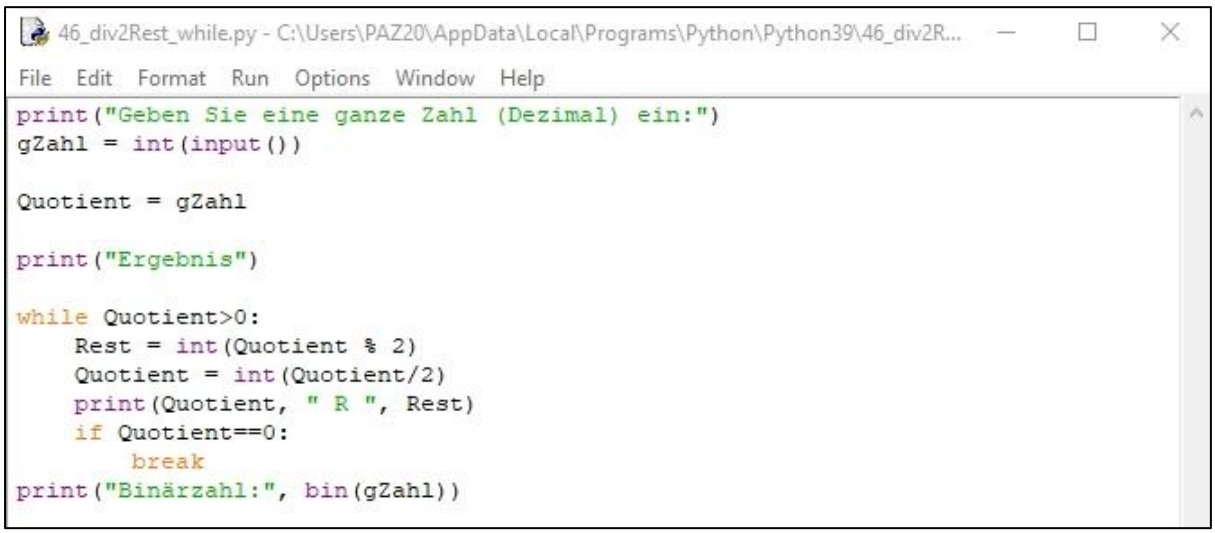

Abb.18: Python-Programm: „Division durch zwei mit Rest“ – mit while-Schleife

Im nächsten Schritt wollen wir eine **eigene Funktion** bilden, aus den Anweisungen in der **while**-Schleife. Dazu wird eine Funktion mit Rückgabewert geschrieben:

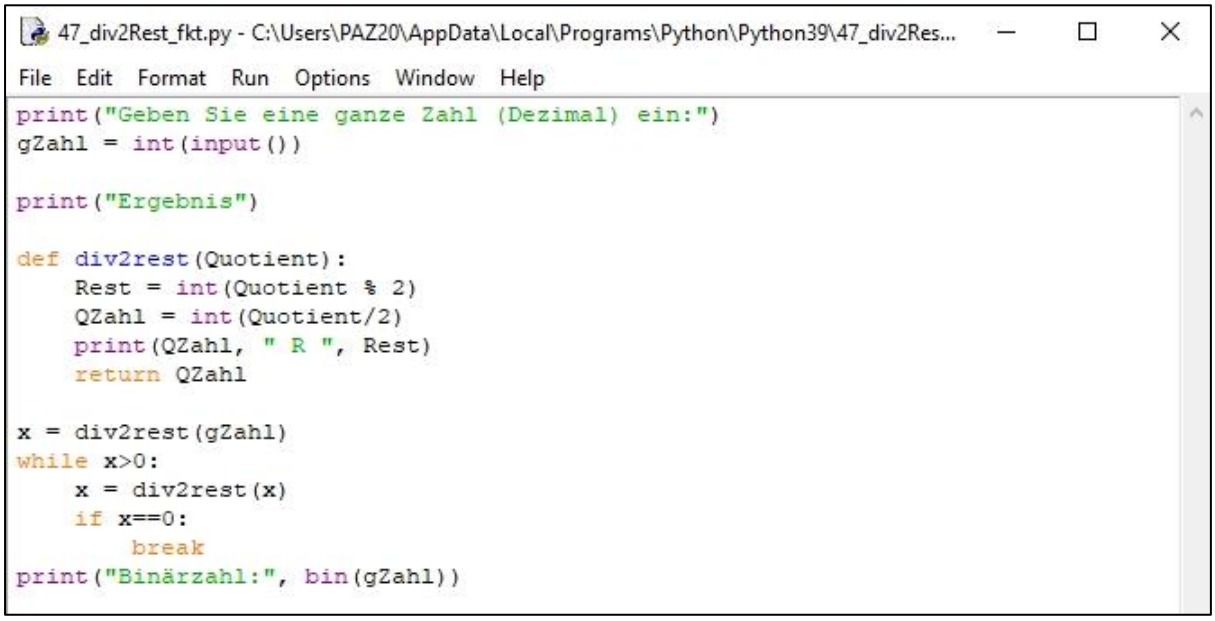

Abb.19: Python-Programm: „Division durch zwei mit Rest“ – Funktion mit Rückgabewert

In der 9. Zeile wird mit dem Ausdruck

9	x = div2rest(gZahl)

die **div2rest()**-Funktion zum ersten Mal aufgerufen und als Rückgabewert erhält man den nächsten Quotienten nach der Division mit 2 (wird hier der Variable **x** zugewiesen). Die **while**-Schleife prüft, ob **x > 0** ist – dann wird die **div2rest()**-Funktion erneut aufgerufen. Ansonsten greift der **if**-Zweig (x = 0) und das Programm wird beendet.

Auch auf die **while**-Schleife können wir durch Schreiben der rekursiven Funktion verzichten.

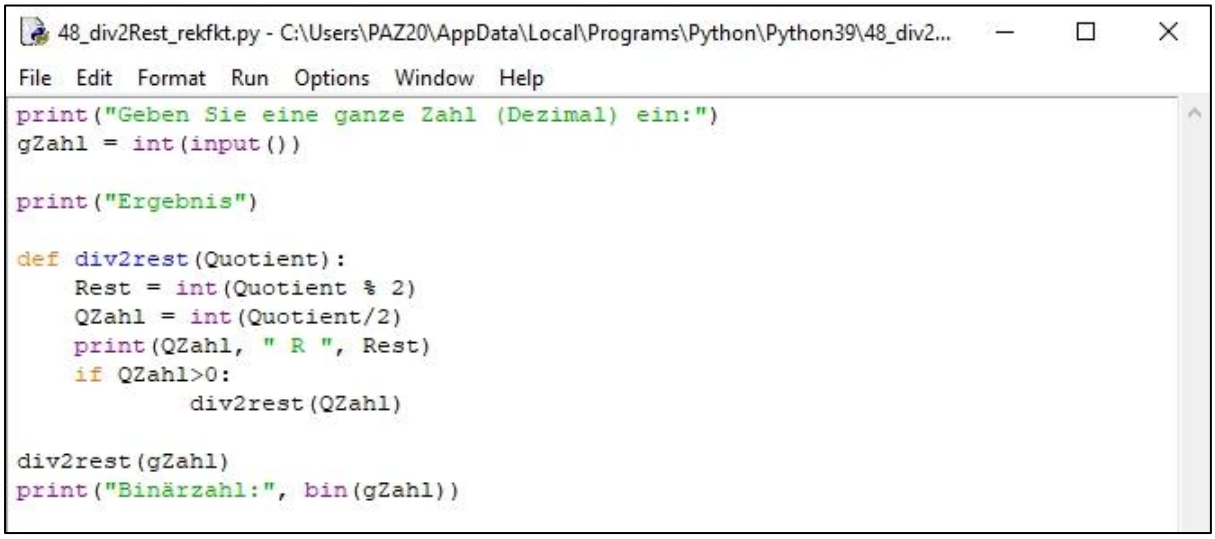

Abb.20: Python-Programm: „Division durch zwei mit Rest“ – rekursive Funktion (ohne while-Schleife)

Eigene Module

Bisher haben wir die eigene Funktion oberhalb des Hauptprogramms in der gleichen Datei definiert. Die eigene Funktion kann einfach in einer separaten Datei gespeichert werden. Der Name der Datei ist zugleich der Name des Moduls.

Module lassen sich einfach mit **import modulname** (= modul dateiname) im Hauptprogramm aufrufen.

Wie in unserem Python-Programm: „Division durch zwei mit Rest“ – Funktion mit Rückgabewert (siehe Abb.: 19) lagern wir die Funktion aus in eine Datei **division2.py**

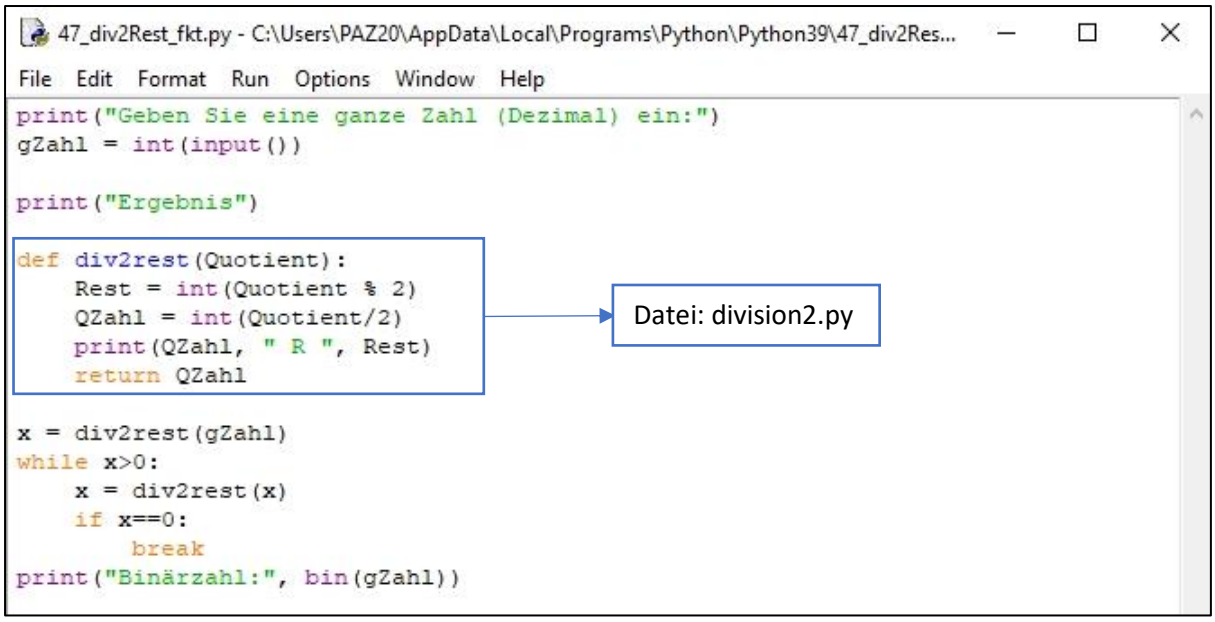

Abb.21: Python-Programm: „Division durch zwei mit Rest“ – Funktion mit Rückgabewert als division2.py

Die **div2rest()**-Funktion aus der Datei „division2.py“ wird über **import division2** aufgerufen.

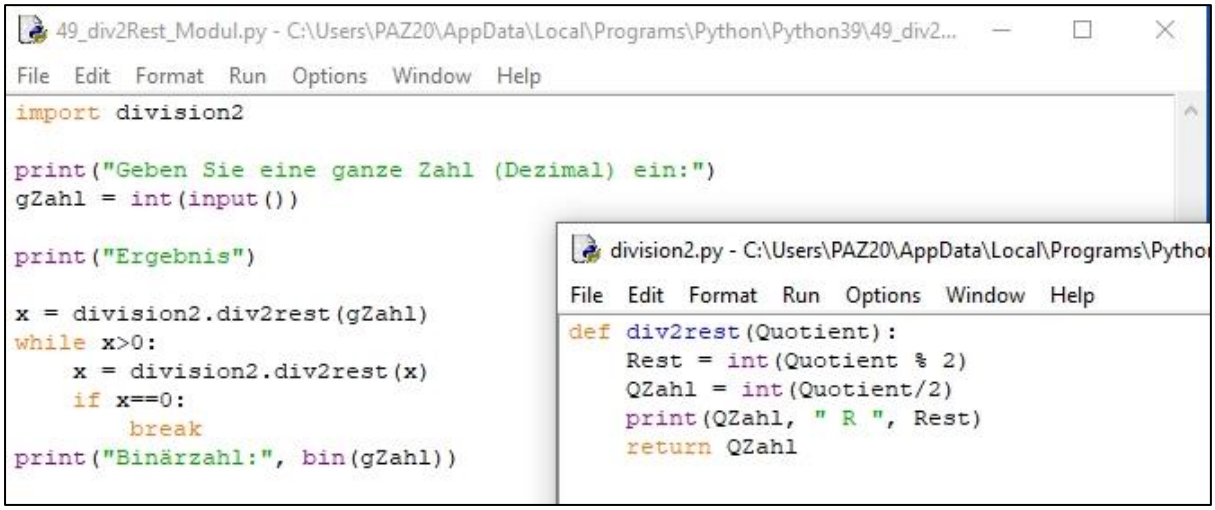

Abb.22: Python-Programm: „Division durch zwei mit Rest“ – import division2 mit der Funktion div2rest()