

20211104_Informationmanagement_FAT1

November 4, 2021

```
[1]: # Funktionen in Python.
# Definition... ist ein Codeblock, der nur ausgeführt wird, wenn er aufgerufen
    ↳ wird.
# Funktionen werden mit dem Schlüsselwort "def" definiert.
# Um eine Funktion aufzurufen, verwenden wir den Funktionsnamen gefolgt von
    ↳ Klammern.

def my_function():
    print('Hello from a function')

my_function()
```

Hello from a function

```
[14]: # Informationen können als Argumente an Funktionen übergeben werden.
# Argumente werden nach Funktionsnamen in Klammern angegeben.
# Wir können beliebig viele Argumente hinzufügen. Getrennt allerdings durch
    ↳ Kommas.

def my_function(fname): #fname ist das Argument
    print(fname + ' Nachname')

my_function('Emil')
my_function('Ann-Kathrin')
```

Emil Nachname

Ann-Kathrin Nachname

```
[8]: # Parameter oder Argumente?
# Die Begriffe können für dasselbe verwendet werden: Informationen die an eine
    ↳ Funktion übergeben werden.

def my_function(fname, lname):
    print(fname + ' ' + lname)

my_function('Bianca', 'Biechele')
my_function('Nino', 'Rudisch')
```

Bianca Biechele
Nino Rudisch

```
[9]: # Um eine Funktion einen Wert zurueckgeben zu lassen, verwenden wir "return"
```

```
def my_function(x):  
    return 5*x  
  
print(my_function(3))
```

15

```
[10]: def my_function(x,y):  
        return x**2+y**2  
  
print(my_function(3,2))
```

13

```
[16]: # Python akzeptiert auch Funktionsrekursion, was bedeutet, dass eine definierte  
      ↪ Funktion sich selbst aufrufen kann.
```

```
# Rekursion ist ein gängiges Konzept. Dies hat den Vorteil, dass wir die Daten  
↪ durchlaufen können, um  
# zu einem Ergebnis zu gelangen.
```

```
# in dem folgenden Beispiel, wird die Funktion "tri-recursion()" definiert,  
↪ welche sich selbst aufruft (recurse).
```

```
# wir verwenden die "k-" Variable als Daten, die bei jeder Wiederholung  
↪ dekrementiert wird (-1).
```

```
# die Rekursion endet, wenn die Bedingung nicht grösser als 0 ist.
```

```
def tri_recursion(k):  
    if(k>0):  
        result = k + tri_recursion(k-1)  
        print(result)  
    else:  
        result = 0  
    return result
```

```
tri_recursion(10)
```

```
# er addiert 1+0=1, dann das Ergebnis 1+2=3, das Ergebnis 3+3=6, das Ergebnis  
↪ 6+4=10, das Ergebnis 10+5=15,...
```

1
3
6

10
15
21
28
36
45
55

[16]: 55

```
[33]: # EOQ Model I -- mit Python

import numpy as np

# Definieren wir die EOQ I Funktion

def EOQ_I(A,D,H):
    # drei Argumente werden definiert
    # A : Vorbereitungskosten der Maschine (setup Kosten)
    # D : Bedarf (jährliche Demand)
    # H : Bestandshaltekosten (Holding Cost)
    # Die Funktion liefert:
    # Q : optimale Bestellmenge
    # Y : optimale Kosten bei der optimalen Bestellmenge
    # D/Q : Optimale Anzahl Bestellungen
    # 12/Anzahl Bestellungen : Zeit zwischen Bestellungen
    # AOC : gesamte jährliche Bestellkosten (annual ordering cost)
    # AHC : gesamte jährliche Bestandshaltekosten (annual holding cost)
    # ATC : gesamte jährliche Kosten

    # Schritt 1. validieren, dass die Parameter alle Positiv sind
    if (A>0 and D>0 and H>0):
        #Schritt 2. EOQ I Modell kalkulieren:
        Q = (np.sqrt(2*A*D/H))
        Y = (np.sqrt(2*A*D*H))
        number_of_orders = D/Q
        time_between_cycles = 12/number_of_orders
        AOC = D/Q*A
        AHC = Q/2*H
        ATC = AOC+AHC
        #Schritt 2. Return eine Liste mit den gewünschten Ergebnissen
        return [Q, Y, number_of_orders, time_between_cycles, AOC, AHC, ATC]
    #Schritt 3. Gebe einen Fehler, wenn die Daten negativ sind.
    else:
        print('Error. Alle Funktionsparameter müssen positiv sein.')

EOQ_I(10, 2400, 0.3)
```

```
[33]: [400.0, 120.0, 6.0, 2.0, 60.0, 60.0, 120.0]
```

```
[25]: # WICHTIG. ;-)  
# Übung. Bitte erstelle eine Funktion, welche EOQ_II und EOQ_III kalkuliert.  
# Nur Q und Y(Q)
```

```
[27]: # Diese Zeile ist nicht Prüfungsrelevant  
  
# nun erstellen wir eine graphische Darstellung des Modells  
# dafür müssen wir zwei Listen generieren für die zwei Achsen: Periode und  
# → Bestand.  
  
# Liste für die Perioden:  
  
period = [0,2]  
while period[-1]<12:  
    period.append(period[-1])  
    period.append(period[-1]+2)  
  
# Liste für den Bestand:  
  
inventory = [400,0]  
while len(inventory)<len(period):  
    inventory.append(400)  
    inventory.append(0)  
  
# plot inventory level  
import matplotlib.pyplot as plt  
  
plt.figure(figsize=(15,5))  
plt.plot(period,inventory)  
plt.xlabel("Month")  
plt.ylabel("Inventory Level")  
plt.title("Economic Order Quantity I")
```

```
[27]: Text(0.5, 1.0, 'Economic Order Quantity I')
```



```
[34]: # Übung: bitte definieren Sie eine Funktion welche die Würzel
      # von  $x^2+y^3+z^3$  berechnet für alle Positive Zahlen.

      # Kondition ist  $x,y,z>0$ 
```

```
[ ]:
```