Python (Funktionen) **Prüfungsstoff**

1. Vorteile des Funktionskonzepts:

   - Wiederverwendbarkeit von Programmcode
   - Zentrale Wartung und Verbesserung von Programmcode
   - Durch Funktionen werden Programmdetails verborgen und somit Programme übersichtlicher bzw. besser lesbar

2. Syntax einer Funktionsdefinition:

```
def myfun(param1, param2, ...):
    funktionsrumpf
```

3. Formale und aktuelle Parameter (Argumente):

```
def funA(x, y, z):
    print(x*y+z)

funA(2,3,4)  # => 10
```

4. Benannte Parameter und Parameterreihenfolge:

```
def funA(x, y, z):
    print(x*y+z)

funA(y=5,z=3,x=8) # => 43
```

5. Default-Werte für Parameter:

```
def funB(x=1, y=1):
    print(x*y+x+y)

funB()     # => 3
funB(5)    # => 11; 'normale' Reihenfolge
funB(y=2)  # => 5
funB(5, 4) # => 29
```

6. Rückgabe von Werten mit `return`.

```
def funC(x, y):
    return 10*x+y

funC(2,3)           # => - [kein print(...) im Funktionsrumpf]
print(funC(2,3))    # => 23
print(2*funC(4,5))  # => 90
```

7. Verschachteln von Funktion:

```
def funD(x, y):
    return x+y

def funE(a):
    return 2*a

x = funD(funE(3), funD(1,2))
print(x)  # => funD(6, 3) => 9
```

8. Gültigkeitsbereich (*scope*) lokaler Variablen

```
def funF(x):
    y = 2*x
    print(y)

y = 8
funF(3)   # => 6
print(y)  # => 8
```

9. Listen als Funktionsparameter:

```
def funG(L):
    L.append(0)

x = [1,2,3]
funG(x)
print(x) # => [1,2,3,0]

y = [4,5,6]
funG(y[:])
print(y) # => [4,5,6]
```