***args** haben 2 Anwendungsfälle:

1. Wenn die Parameterwerte für eine Funktion in Objektform vorliegen (einer Liste oder einem Tupel) können sie eben in dieser Form an die Funktion übergeben werden.
2. Die Anzahl der Übergabeparameter an die Funktion soll variabel bleiben.

1. Übergabe von Parametern in Objektform

- Übergabe der Parameter als Tupel oder Liste. Hier werden die Elemente *In-Order-Of-Appearance* den Parametern der Funktion zugeordnet.

```
def myFunction(a,b,c,*args):
    return print(a,b,c, args)

myFunction(*(1,2,3,4)) # Übergabe als Tupel
myFunction(*[1,2,3,4]) # Übergabe als Liste
>>1, 2, 3 (4,)
```

In-Order-Of-Appearence wurden die Werte 1,2,3 den Parametern a,b,c zugeordnet. Der verbleibende Wert 4, verbleibt ungenutzt als Tupel innerhalb der Funktion und wird über den args-Parameternamen ansprechbar.

Durch den Funktionsaufruf mit *args verkürzt sich gegenüber einem Aufruf mit Parameternamen auch der Quellcode:

```
param = [1,2,3]
myFunction(*param) # Aufruf mit args
myFunction(a=1,b=2,c=3) # Aufruf mit Parameternamen
```

2. Variable Länge der Übergabeparameter

Eine weitere Option die durch den *args-Parameter entsteht, ist die beliebige Fortschreibung von Übergaben an die Funktion. Hier werden im ersten Beispiel der Funktion myFunction 7 Werte übergeben. Die Werte 1,2,3 werden den Parametern a,b,c zugeordnet. Die weiteren 4 Werte werden (wie bereits oben gesehen) als Tupel in der Funktion ansprechbar. Im zweiten Beispiel sind die Keyword-Argumente mit Werten besetzt und das *args-Tupel steht so wie es ist, also mit den Werten 4,5,6,7 , in der Funktion zur Verfügung.

```
myFunction(1,2,3,4,5,6,7)
>>1, 2, 3 (4, 5, 6, 7)

myFunction(1,2,3,(4,5,6,7))
>>1 2 3 ((4, 5, 6, 7),)
```

****kwargs**

- Übergabe der Parameter als Dictionary. Über die Keys werden die Werte den Parametern der Funktion zugeordnet.

**kwargs sind verglichen mit *args eine konservative Variante an die Funktion ein Objekt zu übergeben. Da das Objekt ein Dictionary ist, verhindert man eine Falschzuordnung von

Parameter zu Wert wie es durch die *Order-Of-Appearence*-Regel bei der Verwendung *args passieren kann.

```
def myFunction(a,b,c,**kwargs):
    return print(a,b,c, **kwargs)

myDict = {'b':2,'a':1,'c':3}
myFunction(**myDict)
```

Bei der Nutzung von **kwargs werden alle benötigten Parameter mit ihren Werten als Key-Value Paare in das Dictionary eingetragen und der Funktion übergeben. Enthält das Dictionary Keys, die nicht den Parametern der Funktion zuzuordnen sind, führt dies zu einem Fehler:

```
def myFunction(a,b,c,**kwargs):
    return print(a,b,c, **kwargs)

myDict = {'b':2,'a':1,'c':3,'d':4}
myFunction(**myDict)
```

TypeError: ‚d‘ is an invalid keyword argument for this function

*args und **kwargs gemeinsam in einer Funktion

Natürlich lassen sich *args und **kwargs auch gemeinsam in einer Funktion verwenden. Wir sehen an dem Beispielcode unten 5 Funktionsaufrufe von myFunction, die in ihrer Definition sowohl *args als auch **kwargs enthält.

```
def myFunction(parameter_1, parameter_2, *args ,**kwargs):
    print(parameter_1)
    print(parameter_2)
    print(*args)


myFunction('A',*(100,200,300))
>>A
>>100
>>(200, 300)

myFunction('A',**{'parameter_2':1000})
>>A
>>1000
>>()

myFunction(parameter_2 = 1000, *('A',))
>>A
>>1000
>>()

myFunction('A',**{'parameter_1':1000,'parameter_2':1000})
```

TypeError: myFunction() got multiple values for argument ‚parameter_1‘

```
myFunction('A',*(1,2,3,),**{'parameter_2':1000})
```

TypeError: myFunction() got multiple values for argument ‚parameter_2‘