

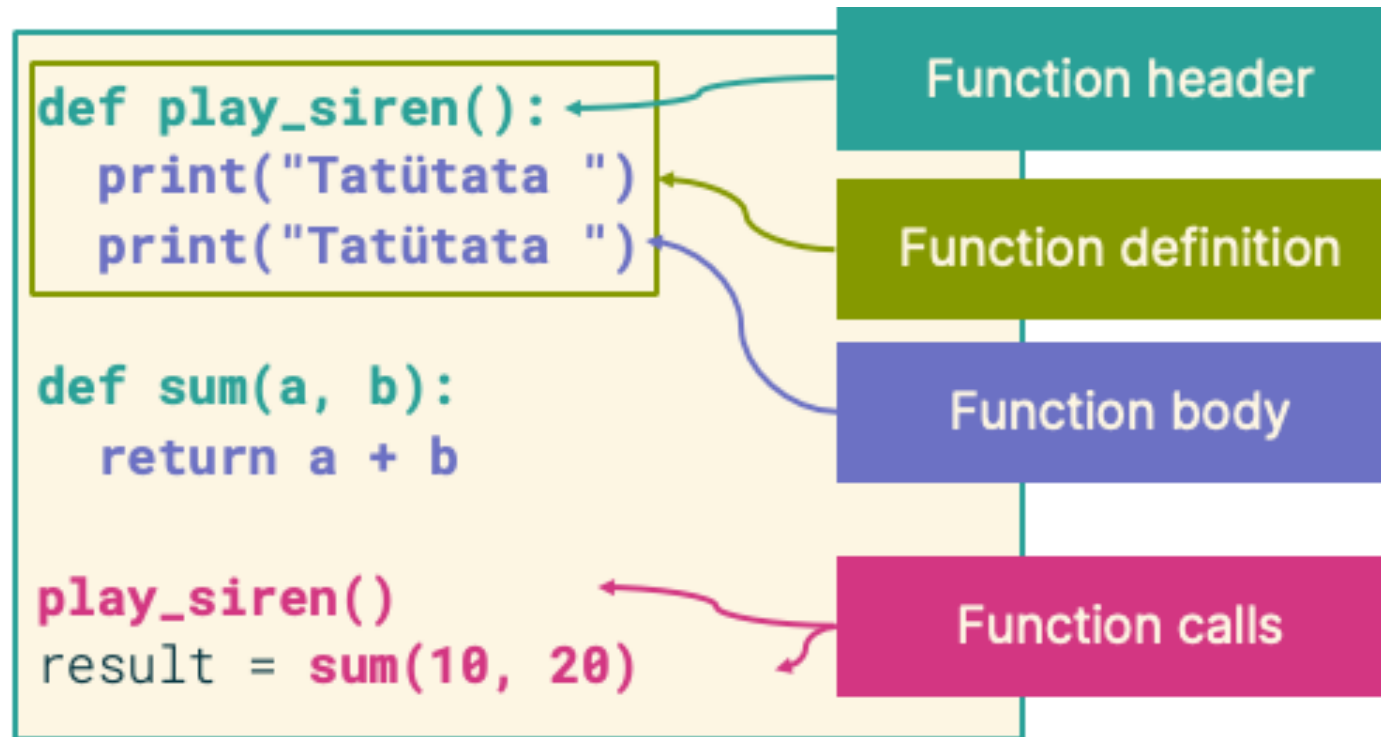
Deep dive Funktionen mit Python

Mehr Modularität in unseren Projekten

Funktionen

In den nächsten Slides werden wir uns nochmals kurz die Funktionen in Erinnerung rufen

Funktionen und Funktionsaufrufe



Funktionen und Funktionsaufrufe

```
1 def play_siren():  
2     print("Tatütata")  
3     print("Tatütata")  
4  
5  
6 def sum(a, b):  
7     return a + b  
8  
9  
10 play_siren()  
11 result = sum(10, 20)
```

Variable Anzahl von Parametern

Bisher wird darauf geachtet, dass die Anzahl der Parameter einer Funktion bei Definition und Aufruf übereinstimmt.

Man kann aber auch Funktionen mit einer **variablen** Anzahl von Parametern definieren.

Variable Anzahl von Parametern

Bei der Definition einer solchen Funktion notieren Sie vor dem letzten (gegebenenfalls einzigen) Parameter das Zeichen *.

Dieser Parameter enthält ein **Tupel** mit den bis dahin nicht zugeordneten Werten der Parameterkette.

Oftmals wird diese Schreibweise auch *args (für arguments) benannt.

Variable Anzahl von Parametern

Hier ein kurzes Beispiel dazu:

```
1 # 16-multi-params.py
2
3 def sumfunc(*params):
4     result = 0
5     for s in params:
6         result += s
7     return result
8
9 print("Sum:", sumfunc(3, 4))
10 print("Sum:", sumfunc(3, 8, 12, -5))
```

Variable Anzahl von Parametern

Hier einige bekannte vertreter für diese Methodik: max(), min(), zip(), print()

```
1 def print(self, *args, sep=' ', end='\n', file=None):
2     """
3     print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
4
5     Prints the values to a stream, or to sys.stdout by default.
6     Optional keyword arguments:
7     file: a file-like object (stream); defaults to the current sys.stdout.
8     sep: string inserted between values, default a space.
9     end: string appended after the last value, default a newline.
10    flush: whether to forcibly flush the stream.
11    """
12    pass
```


Variable Anzahl von Parametern

Es ist auch möglich mittels `**` sogenannte "Key-Value" Pairs zu übergeben.

Bei der Definition einer solchen Funktion notieren Sie vor dem letzten (gegebenenfalls einzigen) Parameter das Zeichen `**`.

Oftmals wird diese Schreibweise auch `**kwargs` (für key-value-arguments) benannt.

Variable Anzahl von Parametern

```
1 def concatenate(**kwargs):
2     result = ""
3     # Iterating over the Python kwargs dictionary
4     for arg in kwargs.values():
5         result += arg
6     return result
7
8 def concatenate_keys(**kwargs):
9     result = ""
10    # Iterating over the keys of the Python kwargs dictionary
11    for arg in kwargs:
12        result += arg
13    return result
14
15 print(concatenate(a="Real", b="Python", c="Is", d="Great", e="!"))
16 print(concatenate_keys(a="Real", b="Python", c="Is", d="Great", e="!"))
```

Variable Anzahl von Parametern

```
1 def concatenate(**kwargs):
2     result = ""
3     # Iterating over the Python kwargs dictionary
4     for arg in kwargs.values():
5         result += arg
6     return result
7
8 def concatenate_keys(**kwargs):
9     result = ""
10    # Iterating over the keys of the Python kwargs dictionary
11    for arg in kwargs:
12        result += arg
13    return result
14
15 print(concatenate(a="Real", b="Python", c="Is", d="Great", e="!"))
16 print(concatenate_keys(a="Real", b="Python", c="Is", d="Great", e="!"))
```

Optionale Parameter

Optionale Parameter ermöglichen ebenfalls eine variable Parameterzahl. Sie müssen einen Vorgabewert besitzen und am Ende der Parameterliste stehen. Zusätzlich können benannte Parameter eingesetzt werden.

```
1 # 16-opt-params.py
2
3 def volume(width, length, depth=1, color="black"):
4     print("Values:", width, length, depth, color)
5     print("Volume:", width * length * depth, "Farbe:", color)
6
7
8 volume(4, 6, 2, "red")
9 volume(2, 12, 7)
10 volume(5, 8)
11 volume(4, 7, color="red")
```

Optionale Parameter

Die beiden Parameter **depth** und **color** sind optional und stehen am Ende der Parameterliste. Es folgen die vier Aufrufe:

- Beim ersten Aufruf werden alle vier Parameter übergeben
- Beim zweiten Aufruf wird nur der erste optionale Parameter übergeben. Der zweite optionale Parameter erhält daher den Vorgabewert
- Beim dritten Aufruf werden beide optionalen Parameter nicht übergeben und erhalten ihren Vorgabewert
- Beim vierten Aufruf wird nur der zweite optionale Parameter übergeben. Da dieser Parameter nicht positionell zugeordnet werden kann, muss er benannt werden

Kopien und Referenzen

Werden Parameter, die an eine Funktion übergeben werden, innerhalb der Funktion verändert, wirkt sich dies unterschiedlich aus:

- Bei der Übergabe eines **einfachen Objekts (Zahl oder Zeichenkette)** wird eine Kopie des Objekts angelegt. Eine Veränderung der Kopie **hat keine Auswirkungen auf das Original**.
- Bei der Übergabe eines Objekts, zum Beispiel **des Typs Liste, Dictionary oder Set**, wird mit einer Referenz auf das **Originalobjekt gearbeitet**. Eine Veränderung über die Referenz **verändert** auch das Original.

Kopien und Referenzen



```
1 # 17-ref-and-copy.py
2
3 def change(za, tx, li, di, st):
4     za = 8
5     tx = "ciao "
6     li[0] = 7
7     di["x"] = 7
8     st.discard(3)
9     print(f"Function: {za} {tx} {li} {di} {st}")
10 hza = 3
11 htx = "hello"
12 hli = [3, "abc"]
13 hdi = {"x":3, "y":"abc"}
14 hst = set([3, "abc"])
15 print(f"Before:   {hza} {htx} {hli} {hdi} {hst}")
16 change(hza, htx, hli, hdi, hst)
17 print(f"After:   {hza} {htx} {hli} {hdi} {hst}")
```

Namespaces

Die Definition einer Funktion in Python erzeugt einen **lokalen Namensraum**. In diesem lokalen Namensraum stehen alle Namen der Variablen, denen innerhalb der Funktion ein Wert zugewiesen wird, und die Namen der Variablen aus der Parameterliste.

```
1 # 17-namespaces.py
2
3 a = "Hello"
4
5 def a_test():
6     a = "Hola"
7     print(a)
8
9 a_test()
10 print(a)
```



Rekursive Funktionen

Bestimmte Abläufe lassen sich am besten rekursiv programmieren.

Eine rekursive Funktion ruft sich immer wieder selbst auf.

Damit dies nicht zu einer endlosen Menge an Funktionsaufrufen führt, muss es eine Bedingung geben, die der Rekursion ein Ende setzt.

Zudem wird ein erster Aufruf benötigt, der die Rekursion einleitet.




```
1 # 17-recursive.py
2
3 def make_half(value):
4     print(value)
5     value = value / 2
6     if value > 0.05:
7         make_half(value)
8 make_half(3)
```

Lambda Funktionen

Lambda-Funktionen werden mithilfe des Schlüsselworts `lambda` erstellt. Sie werden auch **anonyme Funktionen** genannt, im Gegensatz zu den bisher genutzten benannten Funktionen.

Sie haben im Vergleich zu einer benannten Funktion eine **kürzere Definition**.



```
1 # 17-lambda.py
2
3 multiplication = lambda x,y: x*y
4 add = lambda x,y: x+y
5 print(multiplication(5,3))
6 print(add(4,7))
```

Mehrere Rückgabewerte

Python erlaubt auch problemlos mehrere Werte aus einer Funktion zurückzugeben. Dies geschieht im Sinne eines Tuples.

```
1 # 17-multi-return.py
2
3 import math
4 def circle(radius):
5     area = math.pi * radius * radius
6     circumference = 2 * math.pi * radius
7     return area, circumference
8
9 f, u = circle(3)
10 print(f"Area: {round(f,3)}, Circumference: {round(u,3)}")
```

Übung: Funktionen

Wir wollen nun das gelernte nochmals vertiefen.

Bitte schreiben Sie dafür eine Funktion `print_animals(*animals)`, die eine beliebige Anzahl an Tieren (z. B. Haustiere) aufnimmt und diese direkt in der Funktion mittels **`print()`** ausgibt.

Lösung: Funktionen



```
1 # 17-print-animals.py
2
3 def print_animals(*animals):
4     for a in animals:
5         print(a)
6
7 print_animals("Bill", "Tom")
```

Übung: Funktionen

Als zweite kleine Übung soll nun noch eine Funktion geschrieben werden, die verschiedene Merkmale aufnimmt und ausgibt. **Namen**, **Augen-** und **Haarfarbe** müssen eingegeben werden, die **Grösse** und das **Gewicht** sind optional einzugeben (haben einen default-wert).

Nennen Sie die Funktion `print_human_details(...)`

Lösung: Funktionen



```
1 # 17-human-details.py
2
3 def print_human_details(name, eyecolor, haircolor, height=170, weight=65):
4     print("Details: Name", name, "eyecolor", eyecolor, "haircolor",
5           haircolor, "height", height, "weight", weight)
6
7 print_human_details("David", "Brown", "Brown", 171)
```

Übung: Funktionen

Zu guter letzt wollen wir noch eine funktion programmieren mit dem Namen **print_day_month_year(dict)** welche als Parameter ein dictionary erhält {"day": 3, "month": 12, "year": 2023}.

Innerhalb der Funktion, werden 5 Tage addiert, 3 Monate subtrahiert und das Jahr bleibt gleich. Die Funktion soll danach drei Rückgabewerte zurückgeben:

```
return dict["day"], dict["month"], dict["year"]
```

Hiermit soll am Schluss ausserhalb der Funktion der Satz: "Das neue Datum ist der 8.9.2023" gebildet werden.

Lösung: Funktionen

```
1 # 17-multi-return-year.py
2
3 def print_day_month_year(dict):
4     dict["day"] = dict["day"] + 5
5     dict["month"] = dict["month"] - 3
6
7     return dict["day"], dict["month"], dict["year"]
8
9
10 d, m, y = print_day_month_year({"day": 3, "month": 12, "year": 2023})
11
12 print(f"Das neue Datum ist der {d}.{m}.{y}")
```

Ende

Das war alles für dieses Kapitel
