

- Wenn Programmcode wiederverwendet werden soll, müssen wir bisher den entsprechenden Programmteil kopieren.
- Das ist aufwändig, fehleranfällig (bei Änderungen) und erhöht den Umfang des Programms unnötig.
- Deshalb wird in Programmiersprachen das Konzept von **Funktionen** eingeführt.
- Definition einer Funktion in Python:

```
def funktionsname(Parameterliste):  
    anweisung_1  
    ...  
    anweisung_n
```

def ist ein sogenanntes **Schlüsselwort**.

Der Funktionsname ist frei wählbar.

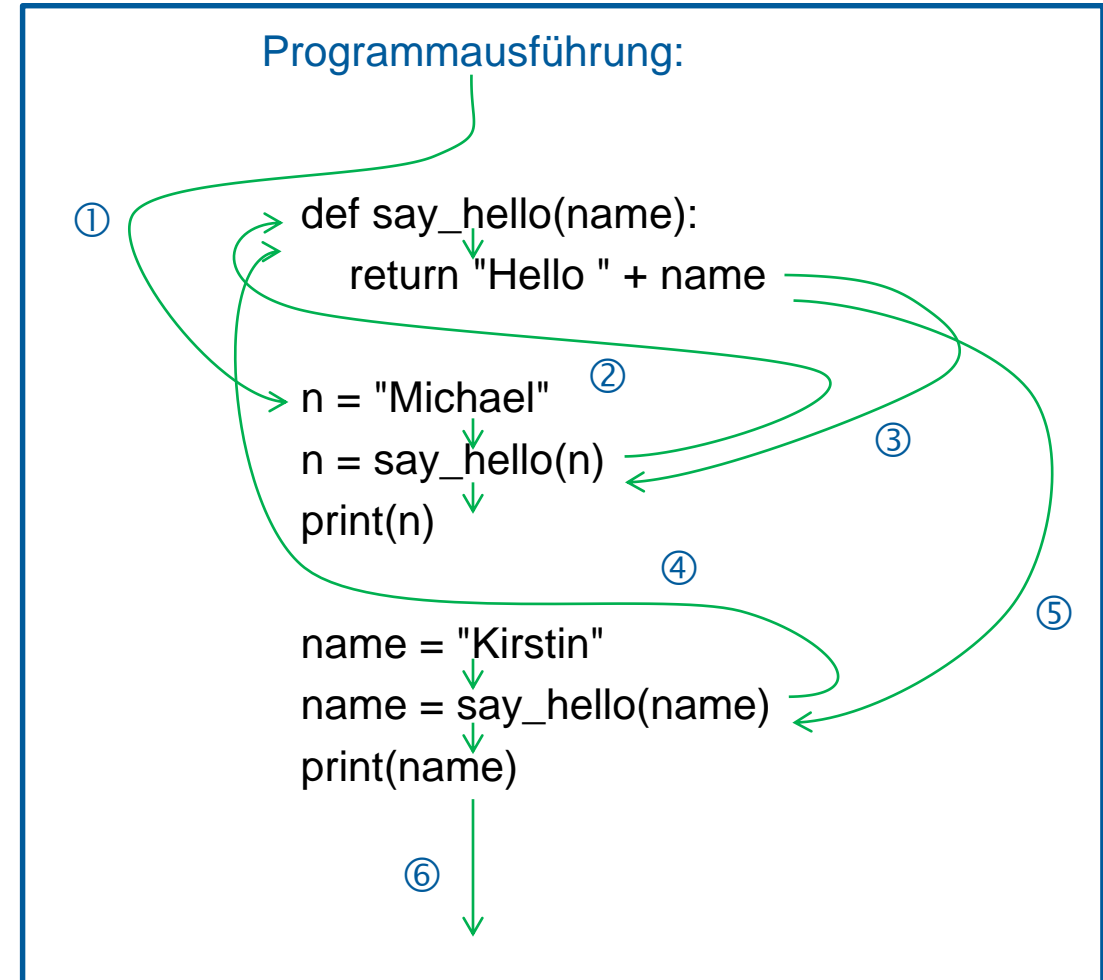
Die gleiche Einrückung der Anweisungen ist unbedingt erforderlich, um die Zugehörigkeit zur Funktion zu kennzeichnen.

■ Beispiel (Klein 2018, S. 114):

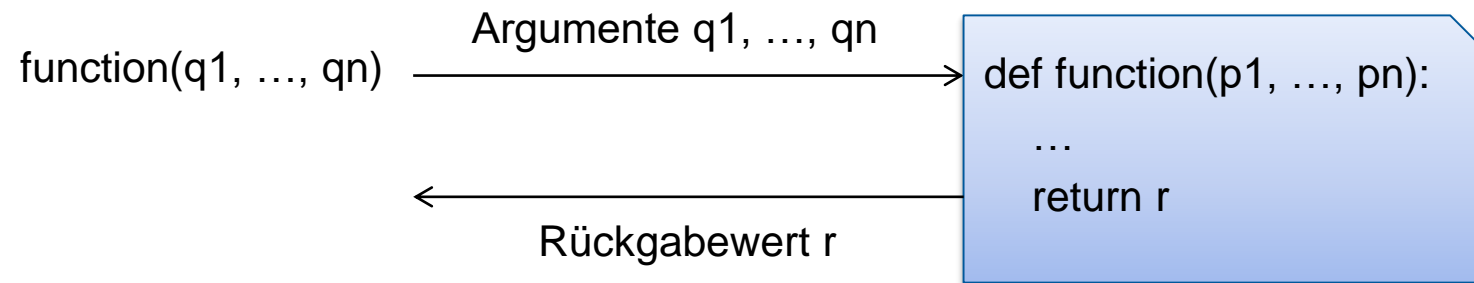
```
def say_hello(name):  
    return "Hello " + name
```

```
n = "Michael"  
n = say_hello(n)  
print(n)                # Ausgabe: Hello Michael
```

```
name = "Kirstin"  
name = say_hello(name)  
print(name)             # Ausgabe: Hello Kirstin
```



- Beim **Aufruf** erhalten Funktionen **Argumente** und berechnen daraus **Rückgabewerte**.



- In der Kopfzeile der Funktion steht eine Liste von **Parametern**. Die Funktion erwartet beim Aufruf für jeden Parameter ein Argument. Die Argumente sind Referenzen auf Werte.
- Die Parameterliste besteht aus durch Kommata getrennten Parameternamen. Die Liste kann auch leer sein.

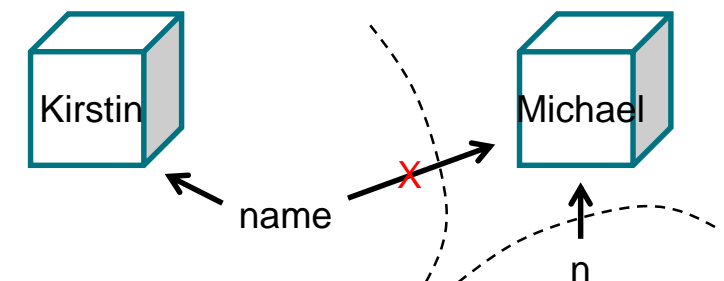
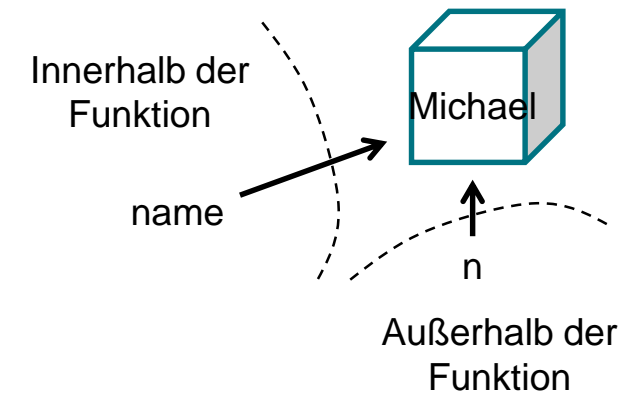
■ Beispiel:

```
def say_hello(name):           # Der Parameter heißt name
    return "Hello " + name
n = "Michael"
n = say_hello(n)               # Das übergebene Argument
                               # ist n und hat den Wert "Michael"
```

Die Zeichenkette "Michael" ist dann über die Referenz *name* in der Funktion verwendbar.

■ Falls dem Parameter ein Wert zugewiesen wird, ändert sich der ursprüngliche Wert nicht. Beispiel:

```
def say_hello(name):
    name = "Kirstin"
    return "Hello " + name
```



- Der Wert des Ausdrucks, der nach return steht, wird zurückgegeben.
- Eine Funktion kann auch mehrere return-Anweisungen an beliebigen Stellen enthalten.
- Nach einer return-Anweisung wird die Funktion verlassen.
- Folgt nach return kein Ausdruck oder wird das Ende der Funktion erreicht, ohne eine return-Anweisung zu durchlaufen, wird der spezielle Wert None zurückgegeben. Diesen Wert kann man auch abfragen.

Beispiel:

```
def funct(n):  
    if n != 0:  
        return n
```

```
r = funct(0)  
if r == None:  
    print("None wurde zurückgegeben")
```

Gleichbedeutend mit:

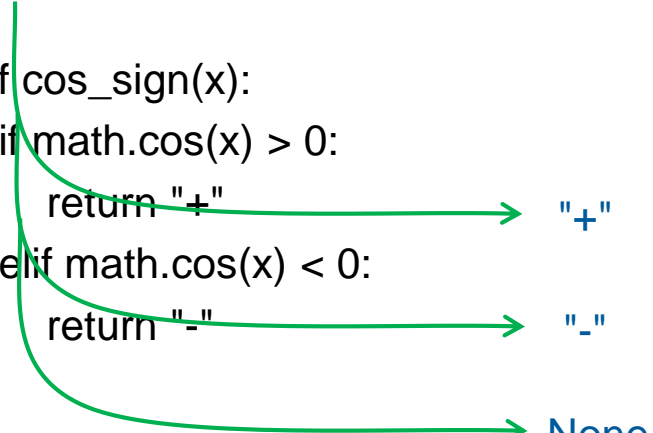
```
def funct(n):  
    if n != 0:  
        return n  
    else:  
        return None
```

■ Welche Konsolenausgabe hat das folgende Skript?

```
def cos_sign(x):  
    if math.cos(x) > 0:  
        return "+"  
    elif math.cos(x) < 0:  
        return "-"  
  
for i in range(0, 35):  
    print(cos_sign(i/10), end=" ")
```

Mögliche Funktionsabläufe:

```
def cos_sign(x):  
    if math.cos(x) > 0:  
        return "+"  
    elif math.cos(x) < 0:  
        return "-"  
    return None
```



- Bitte prägen Sie sich die Bezeichnungsweisen im Zusammenhang mit Funktionen ein:
- Eine Funktion besteht aus einer Kopfzeile und dem Funktionskörper. Der Körper muss in Python einheitlich eingerückt werden.

Kopfzeile im ersten Beispiel:

```
def say_hello(name):
```

Funktionskörper im ersten Beispiel:

```
    return "Hello " + name
```

- In den Klammern stehen die Parameter. Manchmal auch formale Parameter genannt. Im ersten Beispiel: `name`
- Eine Funktion wird aufgerufen durch Angabe des Funktionsnamens und Argumenten. Argumente werden manchmal auch aktuelle Parameter genannt. Beispiel: `say_hello(n)`
- Nach dem Schlüsselwort `return` steht der Rückgabewert. Beispiel: `return "Hello " + name`
- Weil als Argumente Referenzen übergeben werden, nennt man die Art der Parameterübergabe call-by-reference.

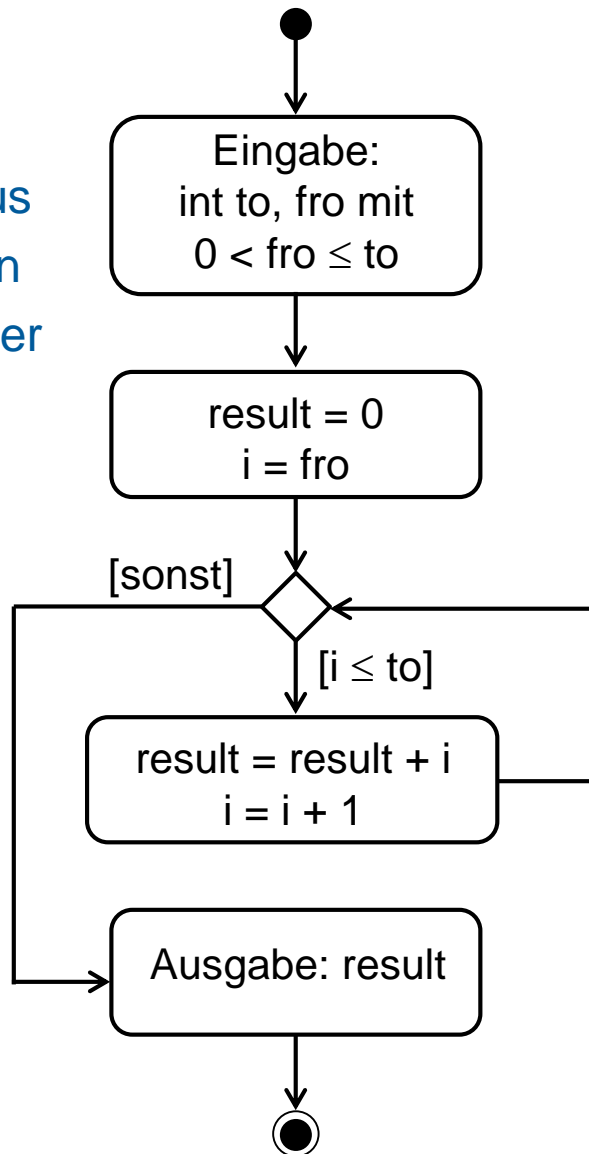
- Vorteile von Funktionen:
 - Wiederverwendung von Programmcode
 - Strukturierung von größeren Programmen
 - Intuitiv verwendbar, da aus der Mathematik bekannt
 - Aufbau von Funktionsbibliotheken möglich.
- Beispiel: Das Modul math enthält u.a. folgende Funktionen:

ceil	factorial	log10	sqrt
cos	floor	log2	tan
degrees	gcd	radians	
exp	log	sin	

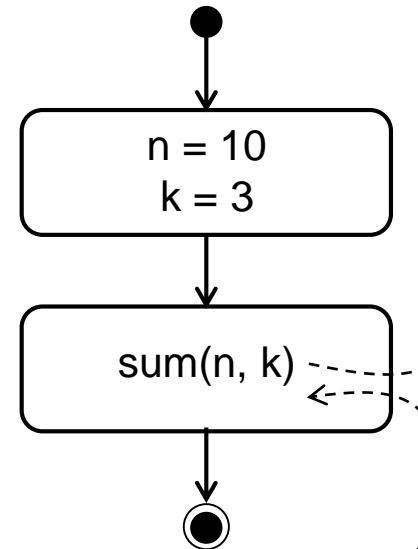
Funktionen (9)



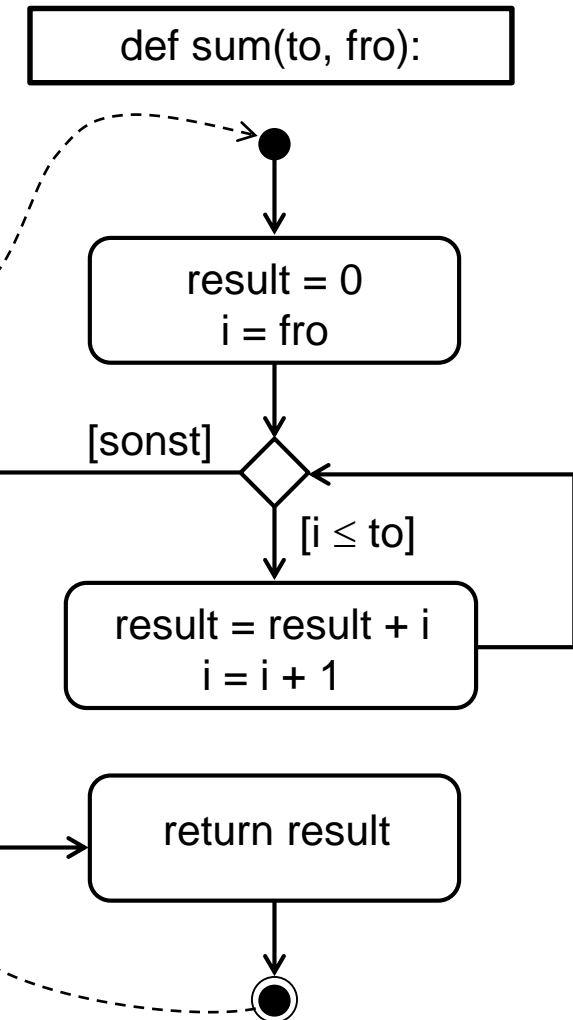
- Beispiel aus dem letzten Kapitel, aber mit zwei Eingaben:



Hauptprogramm:



Als Funktion:



■ Das Beispiel in Python:

```
def sum(to, fro):  
    result = 0  
    i = fro  
    while i <= to:  
        result = result + i  
        i = i + 1  
    return result
```

```
print(sum(10, 3))
```

■ Kapitel 14.1, 14.2, 14.9 in (Klein 2018)

Kürzer geht's mit einer for-Schleife:

```
def sum(to, fro):  
    result = 0  
    for i in range(fro, to+1):  
        result = result + i  
    return result
```

```
print(sum(10, 3))
```



1. Die Programme in den Aufgaben zu den Kapiteln „Schleifen und bedingte Anweisungen in Python“ waren nach dem Schema

Eingabe mit input → Berechnung → Ausgabe mit print

aufgebaut. Formen Sie die Programme der ersten Aufgabe folgendermaßen gleichwertig um:

Funktionsdefinition → Aufruf der Funktion → Ausgabe mit print

Die Argumente sollen dabei fest gewählt werden und nicht mit input eingegeben werden. Die Rückgabewerte sollen mit print ausgegeben werden. Benennen Sie die Funktionen folgendermaßen:

fac, pi_quarter, fib, zafo, third_char, modulo, ggt, max_three, inc_second, frog

Testen Sie die Funktionen.

2. Eine beliebte Übungsaufgabe ist die Prüfung, ob eine Zeichenkette ein Palindrom ist.

Ein Palindrom ist eine Zeichenkette, die von vorne und von hinten gelesen gleich ist. Beispiele sind Anna oder Reittier. Auf Groß- oder Kleinbuchstaben kommt es nicht an. Es können auch Ziffern, Sonderzeichen, Leerzeichen etc. enthalten sein.

Erstellen Sie eine Funktion `is_palindrom(word)`, die prüft, ob `word` ein Palindrom ist. Der Rückgabewert dieser Funktion ist `True`, falls `word` ein Palindrom ist, oder `False`, falls `word` kein Palindrom ist.

Verwenden Sie folgende Idee: Durchlaufen Sie die Zeichenkette in einer for-Schleife mit einer Schleifenvariablen `i` von 0 bis `len(word) // 2` und vergleichen Sie `word[i]` mit `word[length - 1 - i]`.

Zuvor können Sie mit `word = word.lower()` alle Buchstaben in Kleinbuchstaben umwandeln.

Testen Sie die Funktion mit Palindromen und Nicht-Palindromen mit gerader und ungerader Länge

(Klein 2018, Seite 131)

