Namensräume

Namensräume

Namensraum einer Variablen:

Teil des Programmes, wo auf die Variable zugegriffen werden kann

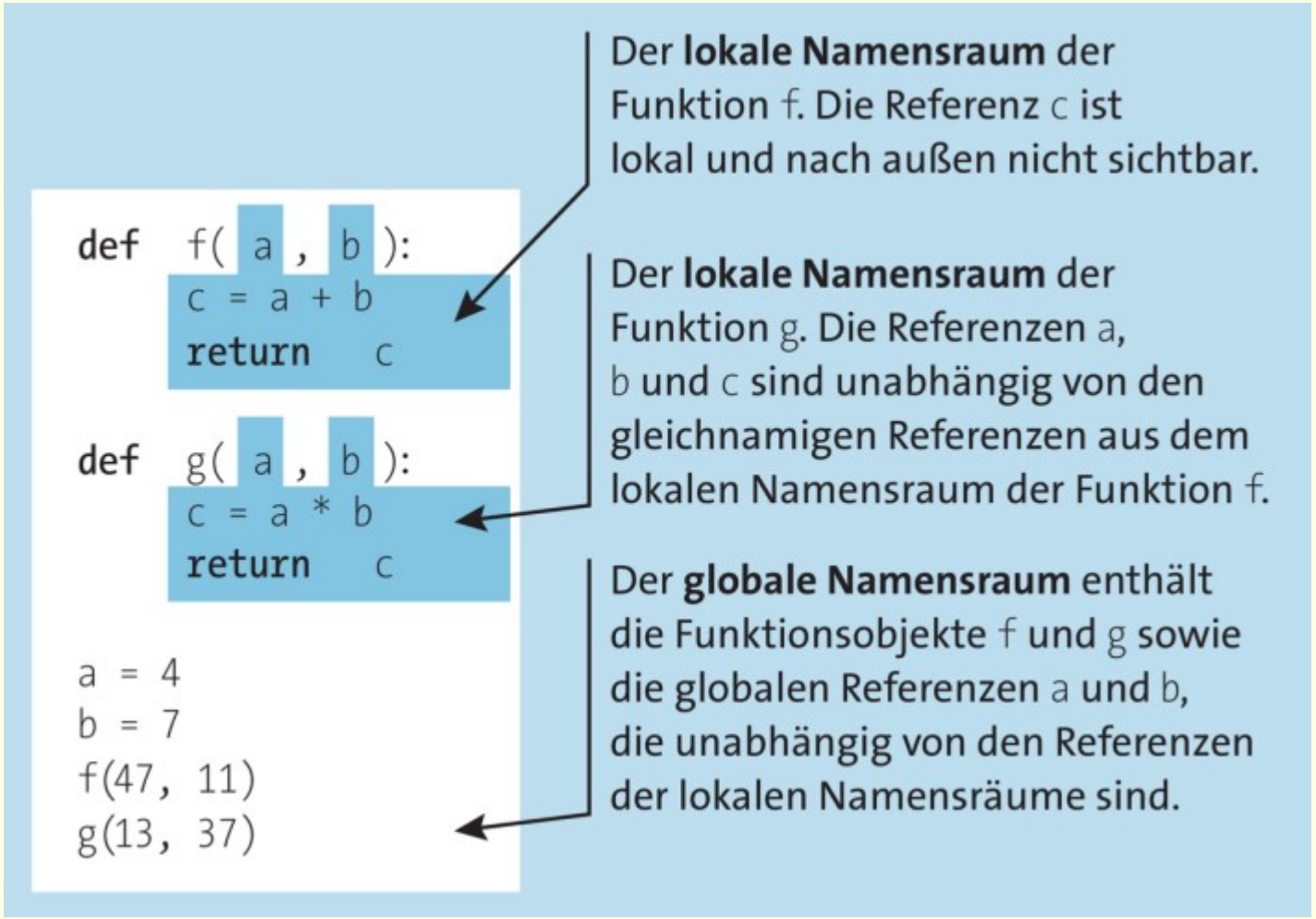
Namensräume

Namensraum einer Variablen:

Teil des Programmes, wo auf die Variable zugegriffen werden kann

- **globale Variablen**: überall verfügbar
- **lokale Variablen**: nur in beschränktem Bereich, z.B. Funktion

Namensräume



Namensräume

Zugriff auf globalen Namensraum

- von überall möglich
- solange Name nicht überlagert
- lokale Version hat Priorität

Namensräume

Lokale Instanzen werden bei Zuweisung erzeugt

```
1 def f():
2     print("vor Zuweisung: " + str(i))
3     i=5
4     print("nach Zuweisung: " + str(i))
5
6 i=1
7 f()

FEHLER
UnboundLocalError: cannot access local variable 'i' where it is not
associated with a value
```

Namensräume

Lokale Instanzen werden bei Zuweisung erzeugt

```
1 def f():
2     print("vor Zuweisung: " + str(i))
3     i=5
4     print("nach Zuweisung: " + str(i))
5
6 i=1
7 f()

FEHLER
UnboundLocalError: cannot access local variable 'i' where it is not
associated with a value
```

Falls Zuweisung, auch davor kein Zugriff auf globale Variable

Namensräume

global verhindert lokale Kopie

```
1 def f():
2     global i
3     print("vor Zuweisung: " + str(i))
4     i=5
5     print("nach Zuweisung: " + str(i))
6
7 i=1
8 f()
9 print(i)

vor Zuweisung: 1
nach Zuweisung: 5
5
```

Namensräume

Auch lokale Funktionen möglich

```
>>> def globale_funktion(n):

...     def lokale_funktion(n):
...         return n**2

...     return lokale_funktion(n)

>>> globale_funktion(2)
4
```

Namensräume

Vorbelegte Funktionsparameter

```
>>> def globale_funktion(n):

...     def lokale_funktion(n):
...         return n**2

...     return lokale_funktion(n)

>>> globale_funktion(2)
4
```

```
>>> def globale_funktion(n):

...     def lokale_funktion(n=n):
...         return n**2

...     return lokale_funktion()

>>> globale_funktion(2)
4
```

Namensräume

Zugriff auf nichtglobale und nichtlokale Variablen

```
>>> def funktion1():
...     def funktion2():
...         nonlocal res
...         res += 1
...     res = 1
...     funktion2()
...     print(res)

>>> funktion1()
2
```

nonlocal sucht "näheste" Variable eines Namens außer globaler

Namensräume

What is the output of the following code?

```
1 num = 1
2 def func():
3     num = 3
4     print(num, end = " ")
5 func()
6 print(num)
7
```

a) The code is erroneus

b) 3 1

c) 1 3

d) 3 3

Anonyme Funktionen

Anonyme Funktionen

Die Sortierfunktion kann Funktion als Parameter nehmen

```
>>> def s(x):
...     return -x

>>> sorted([1,4,7,3,5], key=s)
[7, 5, 4, 3, 1]
```

Anonyme Funktionen

Die Sortierfunktion kann Funktion als Parameter nehmen

- s sehr simpel
- wahrscheinlich nur hier

```
>>> def s(x):
...     return -x

>>> sorted([1,4,7,3,5], key=s)
[7, 5, 4, 3, 1]
```

Anonyme Funktionen

Die Sortierfunktion kann Funktion als Parameter nehmen

```
>>> s = lambda x: -x


>>> sorted([1,4,7,3,5], key=s)
[7, 5, 4, 3, 1]
```

```
>>> def s(x):
...     return -x

>>> sorted([1,4,7,3,5], key=s)
[7, 5, 4, 3, 1]
```

Anonyme Funktionen

Kompakte Definition:

- Schlüsselwort lambda
- Parameterliste
- Doppelpunkt
- arithm. oder log. Ausdruck

```
>>> s = lambda x: -x


>>> sorted([1,4,7,3,5], key=s)
[7, 5, 4, 3, 1]
```

Anonyme Funktionen

Benutzung ohne Zuweisung an Bezeichner

```
>>> sorted([1,4,7,3,5], key=lambda x: -x)
[7, 5, 4, 3, 1]
```

```
>>> s = lambda x: -x

>>> sorted([1,4,7,3,5], key=s)
[7, 5, 4, 3, 1]
```

Anonyme Funktionen

What is the output of the following code?

```
1 f = lambda x: 10
2 print(f(20))
```

```
a) 0
b) 10
c) 20
d) SyntaxError: invalid syntax
```

Anonyme Funktionen

What is the output of the following code?

```
1 func = lambda x: return x
2 print(func(10))
```

```
a) No output
b) 10
c) NameError: name 'x' is not defined
d) SyntaxError: invalid syntax
```

Anonyme Funktionen

What is the output of the following code?

```
>>> sorted(['banana', 'pear', 'grapes', 'apple'], key=lambda x:x[::-1])
```

```
a) ['apple', 'banana', 'grapes', 'pear']

b) ['banana', 'apple', 'pear', 'grapes']

c) SyntaxError: invalid syntax

d) TypeError: 'key' is an invalid keyword argument
for sorted()
```

Anonyme Funktionen

Which option is a valid definition of a lambda assigned to f that adds the parameters x and y?

```
a) f = lambda x, y : x + y

b) f = lambda (x, y):(x + y)

c) f = lambda (x, y): x + y

d) f = lambda x, y : (x + y)
```

Anonyme Funktionen

What is the output of the following code?

```
>>> a, b = 10, 20
>>> (lambda: b, lambda: a)[a < b]()
```

```
a) 10

b) 20

c) TypeError: tuple indices must be integers or slices

d) Invalid Syntax
```

Anonyme Funktionen

What is the output of the following code?

```
1 spam = lambda x, f: x + f(x)
2 print(spam(2, lambda x: x * x), end=' ')
3 print(spam(2, lambda x: x + 3))
```

a) SyntaxError: invalid syntax

b) 6 7

c) 4 5

d) 6 10

Rekursion

Rekursion

Funktionen dürfen sich selbst aufrufen

```
1 def funktion():
2     vari = "abc"
3     funktion()


RecursionError: maximum recursion depth exceeded
```

nur praktische Beschränkung der Tiefe

Rekursion

Funktionen dürfen sich selbst aufrufen

```
>>> from sys import getrecursionlimit

>>> getrecursionlimit()
1000
```

nur praktische Beschränkung der Tiefe

Rekursion

Funktionen dürfen sich selbst aufrufen

```
>>> from sys import getrecursionlimit

>>> getrecursionlimit()
1000

>>> from sys import setrecursionlimit

>>> setrecursionlimit(2000)

>>> getrecursionlimit()
2000
```

Rekursion

Ende der Rekursion mit Abbruchbedingung

```
1  def countdown( k ):
2      print( k , sep = ",", end = "")
3      if k == 0:
4          return              # Abbruchbedingung
5      else:
6          countdown(k - 1)    # Rekursiver Aufruf
7
8  countdown(7)

7,6,5,4,3,2,1,0
```

Rekursion

- evtl. hoher Speicherverbrauch
- Stack muss alle Aufrufe behalten
- viel Verwaltung

Rekursion

Fakultätsfunktion:

fak(n) = n * fak(n-1)

fak(1) = 1

Rekursion

Endrekursion: optimierbar

```
1 def fak(n):
2     if n == 1:
3         return 1
4     else:
5         return n * fak(n-1)
```

Rekursion

Endrekursion: optimierbar

```
1 def fak_endrek(n, partiell=1):
2     if n == 1:
3         return partiell
4     else:
5         return fak_endrek(n-1, partiell * n)
```

Rekursion

Endrekursion: optimierbar

```
1 def fak_endrek(n, partiell=1):
2     if n == 1:
3         return partiell
4     else:
5         return fak_endrek(n-1, partiell * n)
```

gute Übersetzer minimieren den Stack

Rekursion

Die Fibonacci-Folge über den natürlichen Zahlen ist wie folgt definiert:

fib(k) = 0 wenn k=0

fib(k) = 1 wenn k=1

fib(k) = fib(k-1) + fib (k-2) sonst

Schreiben Sie eine Funktion fib(k), die den k-ten Wert der Folge als int zurückliefert.

Rekursion

Herausforderung: Binärsuche

Suche in einer Liste allgemein

- muss alle Elemente anschauen
- soviele Vergleiche wie Elemente

Rekursion

Herausforderung: Binärsuche

Suche in geordneter Liste:

- vergleiche mittleres Element
  - wenn größer, suche in hinterer Hälfte weiter
  - wenn kleiner, suche in vorderer Hälfte weiter
  - wenn gleich: gefunden

Rekursion

Herausforderung: Binärsuche

2	4	5	8	11	12	18	25	29	33	35	36	44	48	55	99

Rekursion

Herausforderung: Binärsuche

Bei rekursiver Umsetzung wichtig:

- nicht vordere/hintere Hälfte der Liste übergeben
- nur Start- und End-Index

OPTIONAL:

Schauen Sie sich den Code für die rekursive Sortiermethode MergeSort an.

https://www.geeksforgeeks.org/python-program-for-merge-sort/

Illustriert komplexere Rekursion und wiederholt Listen.