



Closures

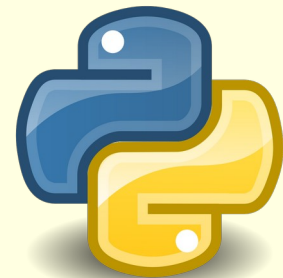
Closures



Variablen können Funktionen enthalten

```
1 def aussen():  
2  
3     def innen():  
4         print("Von innen.")  
5  
6     return "zurück"  
7  
8 func = aussen  
9 print( func() )  
  
zurück  
10
```

Closures

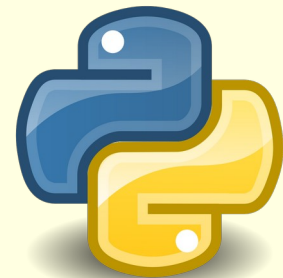


Variablen können Funktionen enthalten

- auch lokale
- auch außerhalb
ihres Namensraumes

```
1 def aussen():  
2  
3     def innen():  
4         print("Von innen.")  
5  
6     return innen  
7  
8 func = aussen()  
9 func()  
  
Von innen.
```

Closures



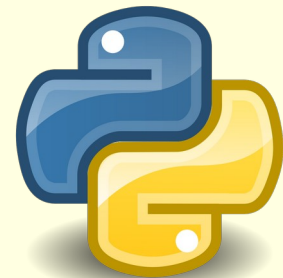
Variablen können Funktionen enthalten

- auch lokale
- mit lokalen Variablen

```
1 def aussen():  
2  
3     def innen():  
4         s = "Von innen"  
5         print( s )  
6  
7     return innen  
8  
9 func = aussen()  
10 func()
```

Von innen

Closures



Variablen können Funktionen enthalten

- lokale Funktion
- mit Variable aus
übergeordnetem
Namensraum

```
1 def aussen():  
2     s = "Von aussen."  
3     def innen():  
4         print(s)  
5  
6     return innen  
7  
8 func = aussen()  
9 func()  
  
???
```

Closures

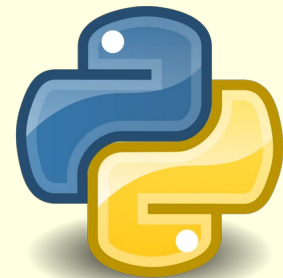


Variablen können Funktionen enthalten

- Inhalt von s ausgegeben
- obwohl s nicht mehr definiert
- Kontext eingeschlossen

```
1 def aussen():  
2     s = "Von aussen."  
3     def innen():  
4         print(s)  
5  
6     return innen  
7  
8 func = aussen()  
9 func()  
  
Von aussen.
```

Closures



Variablen können Funktionen enthalten

- nicht nur lokale Variablen
- auch Parameter der äußeren Funktion

```
1 def aussen(s):  
2  
3     def innen():  
4         print(s)  
5  
6     return innen  
7  
8 func = aussen("Von ganz aussen.")  
9 func()  
  
Von ganz aussen.
```

Closures



Variablen können Funktionen enthalten

- Kontext für jede Instanz
separat eingeschlossen

```
1  def aussen(s):  
2  
3      def innen():  
4          print(s)  
5  
6      return innen  
7  
8  func1 = aussen("aussen 1")  
9  func2 = aussen("aussen 2")  
10 func2()  
11 func1()  
  
aussen 2  
aussen 1
```


Closures

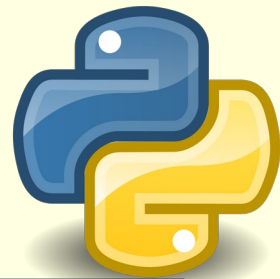


Variablen können Funktionen enthalten

- Hier: jede Instanz von *innen* gibt einen immerselben String zurück

```
1  def aussen(s):  
2  
3      def innen():  
4          print(s)  
5  
6      return innen  
7  
8  func1 = aussen("aussen 1")  
9  func2 = aussen("aussen 2")  
10 func2()  
11 func1()  
  
aussen 2  
aussen 1
```

Closures



Closures und global

- globale Variablen werden nicht eingeschlossen

```
1  w = 100
2
3  def aussen():
4
5      def innen():
6          return w - 1
7
8      return innen
9
10
11 f = aussen()
12 w = 200
13
14 print( f() )
```

199

Closures



Closures und global

- globale Variablen werden nicht eingeschlossen
- auch nicht, wenn lokal verwendet

```
1  w = 100
2
3  def aussen():
4      global w
5      w = 50
6      def innen():
7          return w - 1
8
9      return innen
10
11 f = aussen()
12 w = 200
13
14 print( f() )

199
```



What is the expected outcome of the following code?

```
1 x = 'x'
2 def main():
3     y = 'y'
4     def func():
5         print(x, y, z)
6         return
7     func()
8 z = 'z'
9 if __name__ == "__main__":
10     main()
```

a) No output

b) x y z

c) NameError: name 'z' is not defined

d) SyntaxError: invalid syntax



What is the expected outcome of the following code?

```
1 def x_in(x):  
2     def y_in(y):  
3         return x * y  
4     return y_in  
5  
6 f = x_in(2)  
7 print(f(3))
```

- a) SyntaxError: invalid syntax
- b) 0
- c) NameError: name 'x' is not defined
- d) 6



What is the expected outcome of the following code?

```
1 def f1(x):  
2     y=2  
3     def f2(z):  
4         return x + y + z  
5     return f2  
6  
7 for i in range(3):  
8     func = f1(i)  
9     print(func(i+3), end=' ')
```

- a) 5 7 9
- b) 6 8 10
- c) 567
- d) SyntaxError: invalid syntax
- e) NameError: name z is not defined

Closures



Anwendungen von Closures:

- Factory functions
- Memoisierung
- Funktionen mit überdauerndem Zustand

Closures



Factory functions

- für Funktionen mit mehreren Parametern
- legen einen oder mehrere Parameter fest
- liefern Funktion für Restparameter zurück

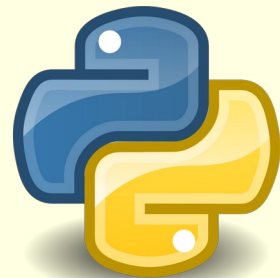
Closures



Factory functions

```
1 def erzeuge_wurzel( wurzelgrad , genauigkeit=2):  
2     def wurzelrechner( zahl ):  
3         return round(pow( zahl, 1 / wurzelgrad), genauigkeit)  
4     return wurzelrechner  
5  
6  
7 quadratwurzel = erzeuge_wurzel(2, 4)  
8 print ( quadratwurzel(25) )  
9  
10 kubikwurzel = erzeuge_wurzel(3, 4)  
11 print ( kubikwurzel(343) )
```

Closures



Factory functions -- **Aufgabe**

Schreiben Sie eine Funktion `intervallerzeuger(a,b)`, die

- eine Funktion mit einem Argument x als Rückgabewert hat, die
- `True` zurückgibt, wenn $a \leq x < b$
- `False` ansonsten

Closures



Memoisierung

- Vermeidung wiederholter Berechnungen
- Speicherung bereits berechneter Werte

Closures



Memoisierung

```
1 def memorahmen():
2     bisherige_werte = {}
3     def berechnung( x ):
4         if x in bisherige_werte:
5             print("nicht neu berechnet")
6             return bisherige_werte[x]
7         else:
8             rueckgabewert = (x*6 + 7) % 23
9             bisherige_werte[x] = rueckgabewert
10            return rueckgabewert
11    return berechnung
12
13 rechnung = memorahmen()
```

Closures



Memoisierung -- **Aufgabe**

Weisen Sie einer Variable eine Funktion mit Parametern x und y zu, die

- $x ** y ** y$ zurückgibt und die
- für jede Parameterkombination den Wert nur einmal berechnet

Closures



Funktionen mit überdauerndem Zustand

- merken sich Zustand nach letztem Aufruf
- Wert in Closure veränderlich
- liefern für selbe Parameter unterschiedliche Werte je nach ihrer Vergangenheit

Closures



Funktionen mit überdauerndem Zustand

```
1 def rahmen ( anzahl ):
2     li = []
3     def runterzaehler():
4         if len( li ) < anzahl:
5             li.append( 'x' )
6             return f"noch {anzahl - len( li )} Aufrufe"
7         else:
8             return "die Funktion ist jetzt verbraucht"
9     return runterzaehler
10
```

Closures



Funktionen mit überdauerndem Zustand

```
1 def rahmen ( anzahl ):  
2     li = []  
3     def runterzaehler():  
4         if len( li ) < anzahl:  
5             li.append( 'x' )  
6             return f"noch {anzahl - len( li )} Aufrufe"  
7         else:  
8             return "die Funktion ist jetzt verbraucht"  
9     return runterzaehler  
10
```

li zeigt bei jedem Aufruf auf selbes Objekt
dessen Wert ändert sich

Closures



Aufgabe

Weisen Sie einer Variable eine Funktion mit einem Parameter n zu, die

- die Summe aller Parameter zurückgibt, die ihr je übergeben wurden.

Closures



Letzte Anwendung

Übergabe eines Parameters an parameterlose Funktion

Beispiel angepasst von

<https://realpython.com/python-closure/#providing-callback-functions>