Scopes in Python (GÃ¼ltigkeitsbereiche) (C) 2017-2021 T.Birnthaler OSTC GmbH
==========

Folgende LEGB-Regel gilt bezÃ¼gliche der REIHENFOLGE der durchsuchten Scopes (GÃ¼ltigkeitsbereiche) bei einem "Name-Lookup" von oben nach unten. Beim ersten Treffer ist die Suche zu Ende:

```
 +------------+-------------------------------------------+  ||
 | L)ocal     | Funktion (auch Lambda)                    |  ||
 | E)nclosing | Einschachtelnde Funktion (z.B. Dekorator) |  ||
 | G)lobal    | Hauptprogramm (Modul)                     |  ||
 | B)uilt-in  | Python-Interpreter (z.B. len, sum, abs)   |  ||
 +------------+-------------------------------------------+  \/
```

ACHTUNG: Es gibt KEINE Block-lokalen Namen in Python (KEINE Block-Scopes, BlÃ¶cke werden durch gleichfÃ¶rmiges EinrÃ¼cken nach einem ":" gebildet. Auch Laufvariablen von for-Schleifen sind global (und enthalten den letzten erreichten Wert):

```
  var = 0                   # var ist globale Variable
  for i in range(1,10+1):   # i ist globale Variable
      j = 10                # j ist globale Variable
      var += i              # var, i sind globale Variable
  print(var, i, j)          # var, i, j sind globale Variable
```

Folgendes Code-StÃ¼ck zeigt dieses LEGB-Verhalten beim "Name-Lookup" (R = Ausgabe-Reihenfolge)

```
 +------------------------+--------------------+---+-------------------------+
 | Code                   | Aktion             | R | Ausgabe                 |
 +------------------------+--------------------+---+-------------------------+
 | print(len)             |                    | 1 | <built-in function len> |
 | def f_aussen():        | Def. Funktion      |   |                         |
 |     len = "E)nclosing" | Def. lok. Var.     |   |                         |
 |     print(len)         |                    | 3 | E)nclosing              |
 |     def f_innen():     | Def. Funktion      |   |                         |
 |         len = "L)okal" | Def. lok. Var.     |   |                         |
 |         print(len)     |                    | 5 | L)ocal                  |
 |     print(len)         |                    | 4 | E)nclosing              |
 |     f_innen()          | Aufruf "f_innen"   |   |                         |
 |     print(len)         |                    | 6 | E)nclosing              |
 | print(len)             |                    | 2 | <built-in function len> |
 | f_aussen()             | Aufruf "f_aussen"  |   |                         |
 | print(len)             |                    | 7 | <built-in function len> |
 | len = "G)lobal"        | Def. glob. Var.    |   |                         |
 | print(len)             |                    | 8 | G)lobal                 |
 | del len                | LÃ¶schen glob. Var. |   |                         |
 | print(len)             |                    | 9 | <built-in function len> |
 +------------------------+--------------------+---+-------------------------+
```

Auf eine GLOBALE Variable kann in Funktionen LESEND zugegriffen werden, falls keine gleichnamige LOKALE Variable durch Zuweisung gebildet wird:

```
  gv = "global"   # Globale Variable gv erzeugen
                  #
  def f(...):     # Funktion f() definieren
      print(gv)   # Globale Variable gv lesen (OK)
                  #
  def g(...):     # Funktion g() definieren
      gv = 123    # Lokale Variable gv erzeugen (OK, gleichnamig)
      print(gv)   # Lokale Variable gv lesen    (OK)
```

Erst eine globale Variable LESEN und dann eine gleichnamige lokale Variable ERZEUGEN ist nicht erlaubt (erzeugt den Fehler "UnboundLocalError" mit der Meldung "local variable ’gv’ referenced before assignment):

```
  def g(...):     # Funktion g() definieren
      print(gv)   # Globale Variable gv lesen   (OK)
      gv = 123    # Lokale Variable gv erzeugen (PENG, gleichnamig)
```

Per SchlÃ¼sselwort "global" kann eine Funktion auf einen GLOBALEN Namen des Hauptprogramms LESEND + SCHREIBEND zugreifen (diese "Schweinerei" wird also durch das SchlÃ¼sselwort "global" deutlich sichtbar gemacht):

```
  gv = "global"       # Globale Variable gv erzeugen
                      #
  def f(...):         # Funktion f() definieren
      global gv       # Globale Variable gv lesen + schreiben erlauben
      print(gv)       # Globale Variable gv lesen     (OK)
      gv = "lokal"    # Globale Variable gv schreiben (OK)
      print(gv)       # Globale Variable gv lesen     (OK)
```

(C) 2025 OSTC GmbH (http://www.ostc.de)

Per SchlÃ¼sselwort "nonlocal" kann eine Funktion auf einen LOKALEN Namen der direkt umgebenden "Wrapper"-Funktion zugreifen (LESEN + SCHREIBEN). Dies ist auch mehrfach bei mehrfach verschachtelten Funktionen mÃ¶glich:

```
  def f(...):              # Funktion f() definieren
      lv = "lokal"         # Lokale Variable lv erzeugen
                           #
      def g(...):          # Eingeschachtelte (lokale) Funktion g() definieren
          nonlocal lv      # Nichtlokale Variable lv von f() verwenden
          print(lv)        # Nichtlokale Variable lv von f() lesen     (OK)
          lv = "lokal"     # Nichtlokale Variable lv von f() schreiben (OK)
          print(lv)        # Nichtlokale Variable lv von f() lesen     (OK)
```