Python les-materialen

# Geneste statements en scope

Nu we onze eigen functies hebben geschreven, is het belangrijk om te begrijpen hoe Python omgaat met de variabelenamen die u toewijst. Wanneer u een variabelenaam in Python aanmaakt, wordt de naam opgeslagen in een *name-space*. Variabelenamen hebben ook een *scope*, de scope bepaalt de zichtbaarheid van die variabelenaam voor andere delen van je code.

Laten we beginnen met een snel gedachte-experiment; stel je de volgende code voor:

x = 25  
  
def printer():  
 x = 50  
 return x  
  
# print(x)  
# print(printer())

Wat denk je dat de uitvoer van printer() is? 25 of 50? Wat is de output van print x? 25 of 50?

print(x)

25

print(printer())

50

Interessant! Maar hoe weet Python naar welke **x** je verwijst in je code? Hier komt het idee van bereik om de hoek kijken. Python heeft een reeks regels die het volgt om te beslissen naar welke variabelen (zoals **x** in dit geval) je verwijst in je code. Laten we de regels opsplitsen:

Dit idee van scope (reikwijdte) in uw code is vooral belangrijk om te begrijpen om namen van variabelen correct toe te wijzen en aan te roepen.

In eenvoudige bewoordingen kan het idee van reikwijdte worden beschreven door 3 algemene regels:

1. Naamtoewijzingen zullen standaard lokale namen maken of wijzigen.
2. Naamreferenties zoeken (maximaal) vier scopes, dit zijn:
   * local (lokaal)
   * enclosing (insluitende) functies
   * globaal
   * (built-in) ingebouwd
3. Namen gedeclareerd in globale en niet-lokale instructies wijzen toegewezen namen toe aan omsluitende (enclosing) module- en functie-scope.

De verklaring in #2 hierboven kan worden gedefinieerd door de LEGB-regel.

**LEGB-regel:**

L: Local (Lokaal) — Namen die op enigerlei wijze binnen een functie zijn toegewezen (def of lambda), en niet globaal zijn gedeclareerd in die functie.

E: Enclosing (Omsluitende) functie locals — Namen in het lokale bereik van alle omsluitende functies (def of lambda), van binnen naar buiten.

G: Globaal (module) — Namen toegewezen op het hoogste niveau van een modulebestand, of globaal gedeclareerd in een definitie in het bestand.

B: Built-in/Ingebouwd (Python) — Namen die vooraf zijn toegewezen in de ingebouwde namenmodule: open, range, SyntaxError,…

## Snelle voorbeelden van LEGB

### Lokaal

# x is local here:  
f = lambda x:x\*\*2

### Enclosing-Functie locals

Dit gebeurt wanneer we een functie binnen een functie hebben (geneste functies)

name = 'This is a global name'  
  
def greet():  
 # Enclosing function  
 name = 'Sammy'  
   
 def hello():  
 print('Hello '+name)  
   
 hello()  
  
greet()

Hello Sammy

Merk op hoe Sammy werd gebruikt, want de hallo()-functie was ingesloten (enclosed) in de greet()-functie!

### Globaal

Gelukkig kun je in Jupyter snel op globale variabelen testen door te kijken of een andere cel de variabele herkent!

print(name)

This is a global name

### Built-in/Ingebouwd

Dit zijn de ingebouwde functienamen in Python (overschrijf deze niet!)

len

<function len>

## Lokale variabelen

Wanneer u variabelen binnen een functiedefinitie declareert, zijn ze op geen enkele manier gerelateerd aan andere variabelen met dezelfde namen die buiten de functie worden gebruikt - d.w.z. de namen van variabelen zijn lokaal voor de functie. Dit wordt het scope/bereik van de variabele genoemd. Alle variabelen hebben het bereik van het blok waarin ze zijn gedeclareerd, te beginnen vanaf het punt van definitie van de naam.

Voorbeeld:

x = 50  
  
def func(x):  
 print('x is', x)  
 x = 2  
 print('Changed local x to', x)  
  
func(x)  
print('x is still', x)

x is 50  
Changed local x to 2  
x is still 50

De eerste keer dat we de waarde van de naam **x** afdrukken met de eerste regel in de hoofdtekst van de functie, gebruikt Python de waarde van de parameter die is gedeclareerd in het hoofdblok, boven de functiedefinitie.

Vervolgens kennen we de waarde 2 toe aan **x**. De naam **x** is lokaal voor onze functie. Dus als we de waarde van **x** in de functie wijzigen, blijft de **x** gedefinieerd in het hoofdblok onaangetast (unaffected).

Met de laatste printopdracht geven we de waarde van **x** weer zoals gedefinieerd in het hoofdblok, waarmee we bevestigen dat deze feitelijk niet wordt beïnvloed door de lokale toewijzing binnen de eerder aangeroepen functie.

## Het global statement

Als je een waarde wilt toewijzen aan een naam die is gedefinieerd op het hoogste niveau van het programma (dus niet binnen een bepaald bereik zoals functies of klassen), dan moet je Python vertellen dat de naam niet lokaal is, maar globaal. We doen dit met behulp van het global statement. Het is onmogelijk om een ​​waarde toe te wijzen aan een variabele die buiten een functie is gedefinieerd zonder de globale instructie.

U kunt de waarden van dergelijke variabelen gebruiken die buiten de functie zijn gedefinieerd (ervan uitgaande dat er geen variabele met dezelfde naam in de functie is). Dit wordt echter niet aangemoedigd (encouraged) en moet worden vermeden, aangezien het voor de lezer van het programma onduidelijk wordt waar de definitie van die variabele is. Het gebruik van de global-instructie maakt het ruimschoots duidelijk dat de variabele is gedefinieerd in een buitenste blok.

Voorbeeld:

x = 50  
  
def func():  
 global x  
 print('This function is now using the global x!')  
 print('Because of global x is: ', x)  
 x = 2  
 print('Ran func(), changed global x to', x)  
  
print('Before calling func(), x is: ', x)  
func()  
print('Value of x (outside of func()) is: ', x)

Before calling func(), x is: 50  
This function is now using the global x!  
Because of global x is: 50  
Ran func(), changed global x to 2  
Value of x (outside of func()) is: 2

De global-instructie wordt gebruikt om te verklaren dat **x** een globale variabele is - dus, wanneer we een waarde toewijzen aan **x** binnen de functie, wordt die verandering weergegeven wanneer we de waarde van **x** in het hoofdblok.

U kunt meer dan één globale variabele specificeren met dezelfde globale instructie, b.v. global x, y, z.

## Conclusie

U zou nu een goed begrip van **scope** moeten hebben (u had intuïtief al gelijk over scope, wat geweldig is!) Een laatste opmerking is dat u de functies **globals()** en **locals()** kunt gebruiken om controleer wat uw huidige lokale en **globale variabelen** zijn.

Een ander ding om in gedachten te houden is dat alles in Python een object is! Ik kan variabelen toewijzen aan functies, net zoals ik dat kan met getallen! We zullen dit nog eens bespreken in het gedeelte decorator (decorateur) van de cursus!