Python les-materialen

# Lambda-expressies, map en filter

Nu is het tijd om snel te leren over twee ingebouwde functies, filter en map. Als we eenmaal leren hoe deze werken, kunnen we meer te weten komen over de lambda-expressie, wat van pas zal komen wanneer je je vaardigheden verder begint te ontwikkelen!

## map functie

Met de functie **map** kunt u een functie “toewijzen” aan een *itereerbaar* object. Dat wil zeggen dat je snel dezelfde functie kunt aanroepen naar elk item in een iterable, zoals een lijst. Bijvoorbeeld:

def square(num):  
 return num\*\*2

my\_nums = [1,2,3,4,5]

map(square,my\_nums)

<map at 0x205baec21d0>

# Om de resultaten te krijgen, itereer je door map() of cast je gewoon naar een lijst  
list(map(square,my\_nums))

[1, 4, 9, 16, 25]

De functies kunnen ook complexer zijn

def splicer(mystring):  
 if len(mystring) % 2 == 0:  
 return 'even'  
 else:  
 return mystring[0]

mynames = ['John','Cindy','Sarah','Kelly','Mike']

list(map(splicer,mynames))

['even', 'C', 'S', 'K', 'even']

## filterfunctie

De filterfunctie retourneert een iterator die items van iterable oplevert (yield) waarvoor functie (item) True is. Dit betekent dat u moet filteren op een functie die True of False retourneert. Geef dat dan door aan het filter (samen met de iterable) en je krijgt alleen de resultaten terug die True zouden retourneren als ze aan de functie werden doorgegeven.

def check\_even(num):  
 return num % 2 == 0

nums = [0,1,2,3,4,5,6,7,8,9,10]

filter(check\_even,nums)

<filter at 0x205baed4710>

list(filter(check\_even,nums))

[0, 2, 4, 6, 8, 10]

## lambda-expressie

Een van de meest bruikbare (en voor beginners, verwarrende) tools van Pythons is de lambda-expressie. lambda-expressies stellen ons in staat om “anonieme” functies te creëren. Dit betekent dat we snel ad-hocfuncties kunnen maken zonder dat we een functie goed hoeven te definiëren met behulp van def.

Functie-objecten die worden geretourneerd door lambda-expressies uit te voeren, werken hetzelfde als die gemaakt en toegewezen door defs. Er is een belangrijk verschil dat lambda nuttig maakt in gespecialiseerde rollen:

**lambda’s body is een enkele expressie, geen blok statements.**

* De body van de lambda is vergelijkbaar met wat we in de return-statement van een def body zouden zetten. We typen het resultaat gewoon als een uitdrukking in plaats van het expliciet terug te geven. Omdat het beperkt is tot een uitdrukking, is een lambda minder algemeen dan een def. We kunnen alleen design knijpen om het nesten van programma’s te beperken. lambda is ontworpen voor het coderen van eenvoudige functies, en def handelt de grotere taken af.

Laten we een lambda-expressie langzaam afbreken door een functie te deconstrueren:

def square(num):  
 result = num\*\*2  
 return result

square(2)

4

We zouden het kunnen vereenvoudigen (simplere manier coderen):

def square(num):  
 return num\*\*2

square(2)

4

We zouden dit zelfs allemaal op één regel kunnen schrijven.

def square(num): return num\*\*2

square(2)

4

Dit is de **vorm van een functie** die een lambda-expressie wil **repliceren**. Een lambda-expressie kan dan worden geschreven als:

lambda num: num \*\* 2

<function \_\_main\_\_.<lambda>>

# Normaal zou je geen naam toewijzen aan een lambda-expressie, dit is alleen voor demonstratie!  
square = lambda num: num \*\*2

square(2)

4

Dus waarom zou je dit gebruiken? Veel functie-aanroepen hebben een functie nodig die wordt doorgegeven, zoals map en filter. Vaak hoeft u de functie die u doorgeeft maar één keer te gebruiken, dus in plaats van deze formeel te definiëren, gebruikt u gewoon de lambda-expressie. Laten we een aantal van de bovenstaande voorbeelden herhalen met een lambda-expressie

list(map(lambda num: num \*\* 2, my\_nums))

[1, 4, 9, 16, 25]

list(filter(lambda n: n % 2 == 0,nums))

[0, 2, 4, 6, 8, 10]

Hier zijn nog een paar voorbeelden. Houd er rekening mee dat hoe complexer een functie is, hoe moeilijker het is om te vertalen in een lambda-expressie, wat betekent dat het soms gewoon eenvoudiger (en vaak de enige manier) is om de def-sleutelwoordfunctie te creëren.

\*\* Lambda-expressie voor het pakken van het eerste teken van een string: \*\*

lambda s: s[0]

<function \_\_main\_\_.<lambda>>

\*\* Lambda-expressie voor het omkeren (reverseren) van een string: \*\*

lambda s: s[::-1]

<function \_\_main\_\_.<lambda>>

U kunt zelfs meerdere argumenten doorgeven aan een lambda-expressie. Houd er rekening mee dat niet elke functie kan worden vertaald in een lambda-expressie.

lambda x,y : x + y

<function \_\_main\_\_.<lambda>>

U zult merken dat u vaak lambda-expressies gebruikt met bepaalde niet-ingebouwde bibliotheken, bijvoorbeeld de panda’s-bibliotheek voor gegevensanalyse werkt heel goed met lambda-expressies.