Python les-materialen

# Functies

## Inleiding tot functies

Deze lezing zal bestaan ​​uit het uitleggen wat een functie is in Python en hoe je er een maakt. Functies zullen een van onze belangrijkste bouwstenen zijn wanneer we steeds grotere hoeveelheden code construeren om problemen op te lossen.

### Wat is een functie?

Formeel is een functie een handig apparaat dat een reeks instructies groepeert, zodat ze meer dan eens kunnen worden uitgevoerd. Ze kunnen ons ook parameters laten specificeren die kunnen dienen als invoer voor de functies.

Op een meer fundamenteel niveau stellen functies ons in staat om niet herhaaldelijk dezelfde code herhaaldelijk te schrijven. Als je je de lessen over strings en lijsten herinnert, onthoud dan dat we een functie len() hebben gebruikt om de lengte van een string te krijgen. Aangezien het controleren van de lengte van een reeks een veelvoorkomende taak is, zou u een functie willen schrijven die dit herhaaldelijk op commando kan doen.

Functies zullen een van de meest basale niveaus zijn van het hergebruik van code in Python, en het zal ons ook in staat stellen om te gaan nadenken over programmaontwerp (we zullen veel dieper ingaan op de ideeën van ontwerp wanneer we leren over objectgeoriënteerd programmeren).

### Waarom zelfs functies gebruiken?

Simpel gezegd, u moet functies gebruiken wanneer u van plan bent een codeblok meerdere keren te gebruiken. Met deze functie kunt u hetzelfde codeblok oproepen zonder het meerdere keren te hoeven schrijven. Dit stelt je op zijn beurt in staat om complexere Python-scripts te maken. Om dit echter echt te begrijpen, moeten we onze eigen functies schrijven!

## Topics van de functies

* def trefwoord (keyword)
* simpele voorbeeld van een functie
* het aanroepen van een functie met ()
* parameters accepteren
* print versus return
* het toevoegen van logica binnen een functie
* meerdere returns binnen een functie
* het toevoegen van lussen (for, while) binnen een functie
* tuple uitpakken
* interacties tussen de functies

### def trefwoord

Laten we eens kijken hoe we de syntaxis van een functie in Python kunnen bouwen. Het heeft de volgende vorm:

def name\_of\_function(arg1,arg2):  
 '''  
 This is where the function's Document String (docstring) goes.  
 When you call help() on your function it will be printed out.  
 '''  
 # Voeg de statements hier toe.  
 # Retourneer het gewenste resultaat

We beginnen met def en dan een **spatie** gevolgd door de naam van de functie. Probeer namen relevant te houden, bijvoorbeeld len() is een goede naam voor een lengte()-functie. Wees ook voorzichtig met namen, je zou een functie niet dezelfde naam willen aanroepen als een [ingebouwde functie in Python](https://docs.python.org/3/library/functions.html) (zoals len).

Vervolgens komen een paar haakjes met verschillende argumenten gescheiden door een **komma**. Deze argumenten zijn de invoer voor uw functie. U kunt deze invoer in uw functie gebruiken en ernaar verwijzen. Hierna zet je een dubbele punt.

Nu is hier de cruciale stap, u moet inspringen om de code in uw functie correct te beginnen. Python maakt gebruik van *whitespace* om code te ordenen. Veel andere programmeertalen doen dit niet, dus houd daar rekening mee.

Vervolgens zie je de docstring, hier schrijf je een basisbeschrijving van de functie. Met Jupyter en Jupyter Notebooks kunt u deze doc-strings lezen door op Shift+Tab te drukken na een functienaam. Doc-strings zijn niet nodig voor eenvoudige functies, maar het is een goede gewoonte om ze in te voeren, zodat u of andere mensen de code die u schrijft gemakkelijk kunnen begrijpen.

Hierna begint u met het schrijven van de code die u wilt uitvoeren.

De beste manier om functies te leren is door voorbeelden door te nemen. Laten we dus proberen voorbeelden door te nemen die betrekking hebben op de verschillende objecten en gegevensstructuren waarover we eerder hebben geleerd.

### Eenvoudig voorbeeld van een functie

def say\_hello():  
 print('hello')

### Het aanroepen van een functie met ()

Roep de functie aan:

say\_hello()

hello

Als u het haakje () vergeet, wordt alleen het feit weergegeven dat say\_hello een functie is. Later zullen we leren dat we functies kunnen doorgeven aan andere functies! Maar onthoud voor nu gewoon om functies aan te roepen met ().

say\_hello

<function \_\_main\_\_.say\_hello>

### Parameters accepteren (argumenten)

Laten we een functie schrijven die mensen begroet met hun naam.

def greeting(name):  
 print(f'Hello {name}')

greeting('Jose')

Hello Jose

## Het gebruiken van return

Tot nu toe hebben we alleen print() gebruikt, maar als we de resulterende variabele willen opslaan, moeten we het sleutelwoord **return** gebruiken.

Laten we enkele voorbeelden bekijken die een return-instructie gebruiken. Met return kan een functie een resultaat *retourneren* dat vervolgens als variabele kan worden opgeslagen of op elke gewenste manier kan worden gebruikt.

### Voorbeeld: Toevoegingsfunctie

def add\_num(num1,num2):  
 return num1+num2

add\_num(4,5)

9

# Kan ook opslaan als variabele vanwege return  
result = add\_num(4,5)

print(result)

9

Wat gebeurt er als we twee strings invoeren?

add\_num('one','two')

'onetwo'

## Veel voorkomende vraag: “Wat is het verschil tussen *return* en *print*?”

**Met het trefwoord return kunt u het resultaat van de uitvoer van een functie daadwerkelijk (actually) als variabele opslaan. De functie print() geeft u eenvoudig de uitvoer weer, maar slaat deze niet op voor toekomstig gebruik. Laten we dit in meer detail onderzoeken**

def print\_result(a,b):  
 print(a+b)

def return\_result(a,b):  
 return a+b

print\_result(10,5)

15

# U zult geen uitvoer zien als u dit in een .py-script uitvoert  
return\_result(10,5)

15

**Maar wat gebeurt er als we dit resultaat echt willen opslaan voor later gebruik?**

my\_result = print\_result(20,20)

40

my\_result

type(my\_result)

NoneType

**Wees voorzichtig! Merk op hoe print\_result() je het resultaat niet in een variabele laat opslaan! Het drukt het alleen af, met print() als resultaat Geen voor de opdracht!**

my\_result = return\_result(20,20)

my\_result

40

my\_result + my\_result

80

# Logica toevoegen aan interne functiebewerkingen (operaties)

Tot nu toe weten we nogal wat over het construeren van logische statements met Python, zoals if/else/elif statements, for en while loops, controleren of een item **in** een lijst is of **niet in** een lijst (Nuttige lezing voor operators). Laten we nu kijken hoe we deze bewerkingen binnen een functie kunnen uitvoeren.

### Controleer of een getal even is

**Herinner de mod-operator % die de rest na deling retourneert, als een getal even is, dan zou mod 2 (% 2) == tot nul moeten zijn.**

2 % 2

0

20 % 2

0

21 % 2

1

20 % 2 == 0

True

21 % 2 == 0

False

\*\* Laten we dit gebruiken om een functie te construeren. Merk op hoe we eenvoudig de booleaanse cheque (boolean) retourneren.\*\*

def even\_check(number):  
 return number % 2 == 0

even\_check(20)

True

even\_check(21)

False

### Controleer of een getal in een lijst even is

Laten we een boolean teruggeven die aangeeft of **any** getal in een lijst even is. Merk hier op hoe **return** uit de lus breekt en de functie verlaat

def check\_even\_list(num\_list):  
 # Go through each number  
 for number in num\_list:  
 # Once we get a "hit" on an even number, we return True  
 if number % 2 == 0:  
 return True  
 # Otherwise we don't do anything  
 else:  
 pass

\*\* Is dit genoeg? NEE! We geven niets terug als ze allemaal oneven zijn!\*\*

check\_even\_list([1,2,3])

True

check\_even\_list([1,1,1])

\*\* ZEER GEMEENSCHAPPELIJKE FOUT!! LATEN WE EEN GEMEENSCHAPPELIJKE LOGISCHE FOUT ZIEN, LET OP DIT IS VERKEERD!!!\*\*

def check\_even\_list(num\_list):  
 # Go through each number  
 for number in num\_list:  
 # Once we get a "hit" on an even number, we return True  
 if number % 2 == 0:  
 return True  
 # This is WRONG! This returns False at the very first odd number!  
 # It doesn't end up checking the other numbers in the list!  
 else:  
 return False

# OH OH! Het geeft False terug na het raken (hitting) van de eerste 1  
check\_even\_list([1,2,3])

False

\*\* Correcte aanpak: we moeten ook een andere return instantieren met False toevoegen die gaat starten NA het doorlopen van de hele lus\*\*

def check\_even\_list(num\_list):  
 # Go through each number  
 for number in num\_list:  
 # Once we get a "hit" on an even number, we return True  
 if number % 2 == 0:  
 return True  
 # Don't do anything if its not even  
 else:  
 pass  
 # Notice the indentation! This ensures we run through the entire for loop   
 return False

check\_even\_list([1,2,3])

True

check\_even\_list([1,3,5])

False

### Retourneer alle even getallen in een lijst

Laten we meer complexiteit toevoegen, we zullen nu alle even getallen in een lijst retourneren, anders een lege lijst.

def check\_even\_list(num\_list):  
   
 even\_numbers = []  
   
 # Go through each number  
 for number in num\_list:  
 # Once we get a "hit" on an even number, we append the even number  
 if number % 2 == 0:  
 even\_numbers.append(number)  
 # Don't do anything if its not even  
 else:  
 pass  
 # Notice the indentation! This ensures we run through the entire for loop   
 return even\_numbers

check\_even\_list([1,2,3,4,5,6])

[2, 4, 6]

check\_even\_list([1,3,5])

[]

## Tuples retourneren voor het uitpakken

\*\* Bedenk dat we een lijst met tuples kunnen doorlopen en de waarden erin kunnen “uitpakken”\*\*

stock\_prices = [('AAPL',200),('GOOG',300),('MSFT',400)]

for item in stock\_prices:  
 print(item)

('AAPL', 200)  
('GOOG', 300)  
('MSFT', 400)

for stock,price in stock\_prices:  
 print(stock)

AAPL  
GOOG  
MSFT

for stock,price in stock\_prices:  
 print(price)

200  
300  
400

**Op dezelfde manier retourneren functies vaak tuples, om gemakkelijk meerdere resultaten te retourneren voor later gebruik.**

Laten we ons de volgende lijst voorstellen:

work\_hours = [('Abby',100),('Billy',400),('Cassie',800)]

De functie beneden wilt werknemer van de maand geeft zowel de naam als het aantal gewerkte uren terug voor de top-presteerder (beoordeeld op het aantal gewerkte uren).

def employee\_check(work\_hours):  
   
 # Set some max value to intially beat, like zero hours  
 current\_max = 0  
 # Set some empty value before the loop  
 employee\_of\_month = ''  
   
 for employee,hours in work\_hours:  
 if hours > current\_max:  
 current\_max = hours  
 employee\_of\_month = employee  
 else:  
 pass  
   
 # Notice the indentation here  
 return (employee\_of\_month,current\_max)

employee\_check(work\_hours)

('Cassie', 800)

## Interacties tussen functies

Functies gebruiken vaak resultaten van andere functies, laten we een eenvoudig voorbeeld bekijken door middel van een raadspel. Er zullen 3 posities in de lijst zijn, waarvan er één een ‘O’ is, een functie zal de lijst schudden, een andere zal de gok van een speler nemen, en ten slotte zal een ander controleren of het juist is. Dit is gebaseerd op het klassieke carnavalsspel waarbij je moet raden onder welke beker een rode bal zit.

**Hoe een lijst in Python te shufflen**

example = [1,2,3,4,5]

from random import shuffle

# Note shuffle is in-place  
shuffle(example)

example

[3, 1, 4, 5, 2]

**OK, laten we ons simpele spel creëeren**

mylist = [' ','O',' ']

def shuffle\_list(mylist):  
 # Lijst opnemen en shuffle-versie teruggeven  
 shuffle(mylist)  
   
 return mylist

mylist

[' ', 'O', ' ']

shuffle\_list(mylist)

[' ', ' ', 'O']

def player\_guess():  
   
 guess = ''  
   
 while guess not in ['0','1','2']:  
   
 # Terugroepen input() retourneert een string  
 guess = input("Pick a number: 0, 1, or 2: ")  
   
 return int(guess)

player\_guess()

Pick a number: 0, 1, or 2: 1  
  
  
  
  
  
1

Nu gaan we de gok van de gebruiker controleren. Merk op dat we hier alleen afdrukken, omdat we de gok van een gebruiker of de geschudde lijst niet hoeven op te slaan.

def check\_guess(mylist,guess):  
 if mylist[guess] == 'O':  
 print('Correct Guess!')  
 else:  
 print('Wrong! Better luck next time')  
 print(mylist)

Nu maken we een beetje setup-logica om alle functies uit te voeren. Merk op hoe ze met elkaar omgaan!

# Oorspronkelijke lijst  
mylist = [' ','O',' ']  
  
# Schud het  
mixedup\_list = shuffle\_list(mylist)  
  
# Krijg een schatting (guess) van de gebruiker  
guess = player\_guess()  
  
# Controleer de gok van de gebruiker  
#------------------------  
# Merk op hoe deze functie de invoer opneemt op basis van de uitvoer van andere functies!  
check\_guess(mixedup\_list,guess)

Pick a number: 0, 1, or 2: 1  
Wrong! Better luck next time  
[' ', ' ', 'O']

Geweldig! U zou nu een basiskennis moeten hebben van het maken van uw eigen functies om uzelf te behoeden voor het herhaaldelijk schrijven van code!