# Oplossingen bij de oefeningen

In dit bestand vind je de oplossingen van een aantal oefeningen. Let erop dat je niet te snel de oplossingen bekijkt. Zorg ervoor dat je jezelf genoeg tijd geeft om de oefeningen op te lossen. Daarnaast zijn er vaak meerdere mogelijke oplossingen en dit bestand toont slecht een manier.

De oplossingen voor de oefeningen van hoofdstuk 12-14 en 18-20 zijn niet toegevoegd, want de oefeningen voor deze hoofdstukken zijn op zichzelf staande projecten. Als je problemen hebt met een oefening uit een van die hoofdstukken, overweeg dan een bericht te zetten op Stack Overflow of r/learnpython.

## 2-2: Simple Messages

```
msg = "I love learning to use Python."
print(msg)

msg = "It's really satisfying!"
print(msg)
```

Output:

```
I love learning to use Python.
It's really satisfying!
```

## 2-5: Beroemde quote

```
print('Albert Einstein once said, "A person who never made a mistake')
print('never tried anything new."')
```

Output:

```
Albert Einstein once said, "A person who never made a mistake
never tried anything new."
```

## 2-7: Witruimte bij namen weghalen

```
name = "\tEric Matthes\n"

print("Unmodified:")
print(name)

print("\nUsing lstrip():")
print(name.lstrip())

print("\nUsing rstrip():")
print(name.rstrip())

print("\nUsing strip():")
print(name.strip())
```

Output:

```
Unmodified:
    Eric Matthes


Using lstrip():
Eric Matthes


Using rstrip():
    Eric Matthes
```

```
Using strip():
Eric Matthes
```

## 2-9: Lievelingsgetal

```
fav_num = 42
msg = "My favorite number is " + str(fav_num) + "."

print(msg)
```

Output:

```
My favorite number is 42.
```

## 3-1: Namen

```
names = ['ron', 'tyler', 'dani']

print(names[0])
print(names[1])
print(names[2])
```

Output:

```
ron
tyler
dani
```

## 3-2: Begroeting

```
names = ['ron', 'tyler', 'dani']

msg = "Hello, " + names[0].title() + "!"
print(msg)

msg = "Hello, " + names[1].title() + "!"
print(msg)

msg = "Hello, " + names[2].title() + "!"
print(msg)
```

Output:

```
Hello, Ron!
Hello, Tyler!
Hello, Dani!
```

## 3-4: Gastenlijst

```
guests = ['guido van rossum', 'jack turner', 'lynn hill']

name = guests[0].title()
print(name + ", please come to dinner.")

name = guests[1].title()
print(name + ", please come to dinner.")

name = guests[2].title()
print(name + ", please come to dinner.")
```

Output:

```
Guido Van Rossum, please come to dinner.
Jack Turner, please come to dinner.
Lynn Hill, please come to dinner.
```

## 3-5: Gastenlijst veranderen

```
# Invite some people to dinner.
guests = ['guido van rossum', 'jack turner', 'lynn hill']

name = guests[0].title()
print(name + ", please come to dinner.")

name = guests[1].title()
print(name + ", please come to dinner.")

name = guests[2].title()
print(name + ", please come to dinner.")

name = guests[1].title()
print("\nSorry, " + name + " can't make it to dinner.")

# Jack can't make it! Let's invite Gary instead.
del(guests[1])
guests.insert(1, 'gary snyder')

# Print the invitations again.
name = guests[0].title()
print("\n" + name + ", please come to dinner.")

name = guests[1].title()
print(name + ", please come to dinner.")

name = guests[2].title()
print(name + ", please come to dinner.")
```

Output:

```
Guido Van Rossum, please come to dinner.
Jack Turner, please come to dinner.
Lynn Hill, please come to dinner.

Sorry, Jack Turner can't make it to dinner.

Guido Van Rossum, please come to dinner.
Gary Snyder, please come to dinner.
Lynn Hill, please come to dinner.
```

## 3-6: Meer gasten

```
# Invite some people to dinner.
guests = ['guido van rossum', 'jack turner', 'lynn hill']

name = guests[0].title()
print(name + ", please come to dinner.")

name = guests[1].title()
print(name + ", please come to dinner.")

name = guests[2].title()
print(name + ", please come to dinner.")

name = guests[1].title()
print("\nSorry, " + name + " can't make it to dinner.")
```

```
# Jack can't make it! Let's invite Gary instead.
del(guests[1])
guests.insert(1, 'gary snyder')

# Print the invitations again.
name = guests[0].title()
print("\n" + name + ", please come to dinner.")

name = guests[1].title()
print(name + ", please come to dinner.")

name = guests[2].title()
print(name + ", please come to dinner.")

# We got a bigger table, so let's add some more people to the list.
print("\nWe got a bigger table!")
guests.insert(0, 'frida kahlo')
guests.insert(2, 'reinhold messner')
guests.append('elizabeth peratrovich')

name = guests[0].title()
print(name + ", please come to dinner.")

name = guests[1].title()
print(name + ", please come to dinner.")

name = guests[2].title()
print(name + ", please come to dinner.")

name = guests[3].title()
print(name + ", please come to dinner.")

name = guests[4].title()
print(name + ", please come to dinner.")

name = guests[5].title()
print(name + ", please come to dinner.")
```

Output:

```
Guido Van Rossum, please come to dinner.
Jack Turner, please come to dinner.
Lynn Hill, please come to dinner.

Sorry, Jack Turner can't make it to dinner.

Guido Van Rossum, please come to dinner.
Gary Snyder, please come to dinner.
Lynn Hill, please come to dinner.

We got a bigger table!
Frida Kahlo, please come to dinner.
Guido Van Rossum, please come to dinner.
Reinhold Messner, please come to dinner.
Gary Snyder, please come to dinner.
Lynn Hill, please come to dinner.
Elizabeth Peratrovich, please come to dinner.
```

## 3-7: Gastenlijst verkleinen

```
# Invite some people to dinner.
guests = ['guido van rossum', 'jack turner', 'lynn hill']

name = guests[0].title()
print(name + ", please come to dinner.")

name = guests[1].title()
print(name + ", please come to dinner.")
```

```
name = guests[2].title()
print(name + ", please come to dinner.")

name = guests[1].title()
print("\nSorry, " + name + " can't make it to dinner.")

# Jack can't make it! Let's invite Gary instead.
del(guests[1])
guests.insert(1, 'gary snyder')

# Print the invitations again.
name = guests[0].title()
print("\n" + name + ", please come to dinner.")

name = guests[1].title()
print(name + ", please come to dinner.")

name = guests[2].title()
print(name + ", please come to dinner.")

# We got a bigger table, so let's add some more people to the list.
print("\nWe got a bigger table!")
guests.insert(0, 'frida kahlo')
guests.insert(2, 'reinhold messner')
guests.append('elizabeth peratrovich')

name = guests[0].title()
print(name + ", please come to dinner.")

name = guests[1].title()
print(name + ", please come to dinner.")

name = guests[2].title()
print(name + ", please come to dinner.")

name = guests[3].title()
print(name + ", please come to dinner.")

name = guests[4].title()
print(name + ", please come to dinner.")

name = guests[5].title()
print(name + ", please come to dinner.")

# Oh no, the table won't arrive on time!
print("\nSorry, we can only invite two people to dinner.")

name = guests.pop()
print("Sorry, " + name.title() + " there's no room at the table.")

name = guests.pop()
print("Sorry, " + name.title() + " there's no room at the table.")

name = guests.pop()
print("Sorry, " + name.title() + " there's no room at the table.")

name = guests.pop()
print("Sorry, " + name.title() + " there's no room at the table.")

# There should be two people left. Let's invite them.
name = guests[0].title()
print(name + ", please come to dinner.")

name = guests[1].title()
print(name + ", please come to dinner.")

# Empty out the list.
del(guests[0])
del(guests[0])
```

```
# Prove the list is empty.
print(guests)
```

Output:

```
Guido Van Rossum, please come to dinner.
Jack Turner, please come to dinner.
Lynn Hill, please come to dinner.

Sorry, Jack Turner can't make it to dinner.

Guido Van Rossum, please come to dinner.
Gary Snyder, please come to dinner.
Lynn Hill, please come to dinner.

We got a bigger table!
Frida Kahlo, please come to dinner.
Guido Van Rossum, please come to dinner.
Reinhold Messner, please come to dinner.
Gary Snyder, please come to dinner.
Lynn Hill, please come to dinner.
Elizabeth Peratrovich, please come to dinner.

Sorry, we can only invite two people to dinner.
Sorry, Elizabeth Peratrovich there's no room at the table.
Sorry, Lynn Hill there's no room at the table.
Sorry, Gary Snyder there's no room at the table.
Sorry, Reinhold Messner there's no room at the table.
Frida Kahlo, please come to dinner.
Guido Van Rossum, please come to dinner.
[]
```

## 3-8: De wereld zien

```
locations = ['himalaya', 'andes', 'tierra del fuego', 'labrador', 'guam']

print("Original order:")
print(locations)

print("\nAlphabetical:")
print(sorted(locations))

print("\nOriginal order:")
print(locations)

print("\nReverse alphabetical:")
print(sorted(locations, reverse=True))

print("\nOriginal order:")
print(locations)

print("\nReversed:")
locations.reverse()
print(locations)

print("\nOriginal order:")
locations.reverse()
print(locations)

print("\nAlphabetical")
locations.sort()
print(locations)

print("\nReverse alphabetical")
locations.sort(reverse=True)
print(locations)
```

Output:

```
Original order:
['himalaya', 'andes', 'tierra del fuego', 'labrador', 'guam']

Alphabetical:
['andes', 'guam', 'himalaya', 'labrador', 'tierra del fuego']

Original order:
['himalaya', 'andes', 'tierra del fuego', 'labrador', 'guam']

Reverse alphabetical:
['tierra del fuego', 'labrador', 'himalaya', 'guam', 'andes']

Original order:
['himalaya', 'andes', 'tierra del fuego', 'labrador', 'guam']

Reversed:
['guam', 'labrador', 'tierra del fuego', 'andes', 'himalaya']

Original order:
['himalaya', 'andes', 'tierra del fuego', 'labrador', 'guam']

Alphabetical
['andes', 'guam', 'himalaya', 'labrador', 'tierra del fuego']

Reverse alphabetical
['tierra del fuego', 'labrador', 'himalaya', 'guam', 'andes']
```

## 4-1: Pizza's

```
favorite_pizzas = ['pepperoni', 'hawaiian', 'veggie']

# Print the names of all the pizzas.
for pizza in favorite_pizzas:
    print(pizza)

print("\n")

# Print a sentence about each pizza.
for pizza in favorite_pizzas:
    print("I really love " + pizza + " pizza!")

print("\nI really love pizza!")
```

Output:

```
pepperoni
hawaiian
veggie


I really love pepperoni pizza!
I really love hawaiian pizza!
I really love veggie pizza!

I really love pizza!
```

## 4-3: Tot twintig tellen

```
numbers = list(range(1, 21))

for number in numbers:
    print(number)
```

Output:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```

## 4-5: Eén miljoen optellen

```
numbers = list(range(1, 1000001))

print(min(numbers))
print(max(numbers))
print(sum(numbers))
```

Output:

```
1
1000000
500000500000
```

## 4-7: Veelvouden van 3

```
threes = list(range(3, 31, 3))

for number in threes:
    print(number)
```

Output:

```
3
6
9
12
15
18
21
24
27
30
```

## 4-8: Tot de derde macht

```
cubes = []
for number in range(1, 11):
    cube = number**3
    cubes.append(cube)

for cube in cubes:
```

```
    print(cube)
```

Output:

```
1
8
27
64
125
216
343
512
729
1000
```

## 4-9: Tot de derde macht met lijstcomprehensie

```
cubes = [number**3 for number in range(1,11)]

for cube in cubes:
    print(cube)
```

Output:

```
1
8
27
64
125
216
343
512
729
1000
```

## 4-11: Mijn pizza's, jouw pizza's

```
favorite_pizzas = ['pepperoni', 'hawaiian', 'veggie']
friend_pizzas = favorite_pizzas[:]

favorite_pizzas.append("meat lover's")
friend_pizzas.append('pesto')

print("My favorite pizzas are:")
for pizza in favorite_pizzas:
    print("- " + pizza)

print("\nMy friend's favorite pizzas are:")
for pizza in friend_pizzas:
    print("- " + pizza)
```

Output:

```
My favorite pizzas are:
- pepperoni
- hawaiian
- veggie
- meat lover's

My friend's favorite pizzas are:
- pepperoni
- hawaiian
- veggie
- pesto
```

## 4-13: Buffet

```
menu_items = (
    'rockfish sandwich', 'halibut nuggets', 'smoked salmon chowder',
    'salmon burger', 'crab cakes',
    )

print("You can choose from the following menu items:")
for item in menu_items:
    print("- " + item)

menu_items = (
    'rockfish sandwich', 'halibut nuggets', 'smoked salmon chowder',
    'black cod tips', 'king crab legs',
    )

print("\nOur menu has been updated.")
print("You can now choose from the following items:")
for item in menu_items:
    print("- " + item)
```

Output:

```
You can choose from the following menu items:
- rockfish sandwich
- halibut nuggets
- smoked salmon chowder
- salmon burger
- crab cakes

Our menu has been updated.
You can now choose from the following items:
- rockfish sandwich
- halibut nuggets
- smoked salmon chowder
- black cod tips
- king crab legs
```

## 5-3: Alien Colors #1

Versie die slaagt:

```
alien_color = 'green'

if alien_color == 'green':
    print("You just earned 5 points!")
```

Output:

```
You just earned 5 points!
```

Versie die niet slaagt:

```
alien_color = 'red'

if alien_color == 'green':
    print("You just earned 5 points!")
```

(geen output)

## 5-4: Alien Colors #2

`if` block runs:

```
alien_color = 'green'

if alien_color == 'green':
    print("You just earned 5 points!")
else:
    print("You just earned 10 points!")
```

Output:

```
You just earned 5 points!
```

`else` block runs:

```
alien_color = 'yellow'

if alien_color == 'green':
    print("You just earned 5 points!")
else:
    print("You just earned 10 points!")
```

Output:

```
You just earned 10 points!
```

## 5-5: Alien Colors #3

```
alien_color = 'red'

if alien_color == 'green':
    print("You just earned 5 points!")
elif alien_color == 'yellow':
    print("You just earned 10 points!")
else:
    print("You just earned 15 points!")
```

Output voor `'red'` alien:

```
You just earned 15 points!
```

## 5-6: Levensfasen

```
age = 17

if age < 2:
    print("You're a baby!")
elif age < 4:
    print("You're a toddler!")
elif age < 13:
    print("You're a kid!")
elif age < 20:
    print("You're a teenager!")
elif age < 65:
    print("You're an adult!")
else:
    print("You're an elder!")
```

Output:

```
You're a teenager!
```

## 5-7: Lievelingsfruit

```
favorite_fruits = ['blueberries', 'salmonberries', 'peaches']

if 'bananas' in favorite_fruits:
    print("You really like bananas!")
if 'apples' in favorite_fruits:
    print("You really like apples!")
if 'blueberries' in favorite_fruits:
    print("You really like blueberries!")
if 'kiwis' in favorite_fruits:
    print("You really like kiwis!")
if 'peaches' in favorite_fruits:
    print("You really like peaches!")
```

Output:

```
You really like blueberries!
You really like peaches!
```

## 5-8: Hallo beheerder

```
usernames = ['eric', 'willie', 'admin', 'erin', 'ever']

for username in usernames:
    if username == 'admin':
        print("Hello admin, would you like to see a status report?")
    else:
        print("Hello " + username + ", thank you for logging in again!")
```

Output:

```
Hello eric, thank you for logging in again!
Hello willie, thank you for logging in again!
Hello admin, would you like to see a status report?
Hello erin, thank you for logging in again!
Hello ever, thank you for logging in again!
```

## 5-9: Geen gebruikers

```
usernames = []

if usernames:
    for username in usernames:
        if username == 'admin':
            print("Hello admin, would you like to see a status report?")
        else:
            print("Hello " + username + ", thank you for logging in again!")
else:
    print("We need to find some users!")
```

Output:

```
We need to find some users!
```

## 5-10: Gebruikersnamen controleren

```
current_users = ['eric', 'willie', 'admin', 'erin', 'Ever']
new_users = ['sarah', 'Willie', 'PHIL', 'ever', 'Iona']

current_users_lower = [user.lower() for user in current_users]

for new_user in new_users:
    if new_user.lower() in current_users_lower:
```

```
        print("Sorry " + new_user + ", that name is taken.")
    else:
        print("Great, " + new_user + " is still available.")
```

Output:

```
Great, sarah is still available.
Sorry Willie, that name is taken.
Great, PHIL is still available.
Sorry ever, that name is taken.
Great, Iona is still available.
```

Let op: als je nog niet vertrouwd bent met lijst comprehensies, kan de lijst `current_users_lower` gemaakt worden door een lus te gebruiken:

```
current_users_lower = []
for user in current_users:
    current_users_lower.append(user.lower())
```

## 5-11: Rangtelwoorden

```
numbers = list(range(1,10))

for number in numbers:
    if number == 1:
        print("1st")
    elif number == 2:
        print("2nd")
    elif number == 3:
        print("3rd")
    else:
        print(str(number) + "th")
```

Output:

```
1st
2nd
3rd
4th
5th
6th
7th
8th
9th
```

## 6-1: Persoon

```
person = {
    'first_name': 'eric',
    'last_name': 'matthes',
    'age': 43,
    'city': 'sitka',
    }

print(person['first_name'])
print(person['last_name'])
print(person['age'])
print(person['city'])
```

Output:

```
eric
matthes
43
sitka
```

## 6-2: Geluksgetallen

```
favorite_numbers = {
    'mandy': 42,
    'micah': 23,
    'gus': 7,
    'hank': 1000000,
    'maggie': 0,
    }

num = favorite_numbers['mandy']
print("Mandy's favorite number is " + str(num) + ".")

num = favorite_numbers['micah']
print("Micah's favorite number is " + str(num) + ".")

num = favorite_numbers['gus']
print("Gus's favorite number is " + str(num) + ".")

num = favorite_numbers['hank']
print("Hank's favorite number is " + str(num) + ".")

num = favorite_numbers['maggie']
print("Maggie's favorite number is " + str(num) + ".")
```

Output:

```
Mandy's favorite number is 42.
Micah's favorite number is 23.
Gus's favorite number is 7.
Hank's favorite number is 1000000.
Maggie's favorite number is 0.
```

## 6-3: Woordenlijst

```
glossary = {
    'string': 'A series of characters.',
    'comment': 'A note in a program that the Python interpreter ignores.',
    'list': 'A collection of items in a particular order.',
    'loop': 'Work through a collection of items, one at a time.',
    'dictionary': "A collection of key-value pairs.",
    }

word = 'string'
print("\n" + word.title() + ": " + glossary[word])

word = 'comment'
print("\n" + word.title() + ": " + glossary[word])

word = 'list'
print("\n" + word.title() + ": " + glossary[word])

word = 'loop'
print("\n" + word.title() + ": " + glossary[word])

word = 'dictionary'
print("\n" + word.title() + ": " + glossary[word])
```

Output:

```
String: A series of characters.

Comment: A note in a program that the Python interpreter ignores.

List: A collection of items in a particular order.

Loop: Work through a collection of items, one at a time.
```

```
Dictionary: A collection of key-value pairs.
```

## 6-4: Woordenlijst 2

```
glossary = {
    'string': 'A series of characters.',
    'comment': 'A note in a program that the Python interpreter ignores.',
    'list': 'A collection of items in a particular order.',
    'loop': 'Work through a collection of items, one at a time.',
    'dictionary': "A collection of key-value pairs.",
    'key': 'The first item in a key-value pair in a dictionary.',
    'value': 'An item associated with a key in a dictionary.',
    'conditional test': 'A comparison between two values.',
    'float': 'A numerical value with a decimal component.',
    'boolean expression': 'An expression that evaluates to True or False.',
    }

for word, definition in glossary.items():
    print("\n" + word.title() + ": " + definition)
```

Output:

```
Dictionary: A collection of key-value pairs.

String: A series of characters.

Boolean Expression: An expression that evaluates to True or False.

Comment: A note in a program that the Python interpreter ignores.

Value: An item associated with a key in a dictionary.

Loop: Work through a collection of items, one at a time.

List: A collection of items in a particular order.

Conditional Test: A comparison between two values.

Key: The first item in a key-value pair in a dictionary.

Float: A numerical value with a decimal component.
```

## 6-5: Rivieren

```
rivers = {
    'nile': 'egypt',
    'mississippi': 'united states',
    'fraser': 'canada',
    'kuskokwim': 'alaska',
    'yangtze': 'china',
    }

for river, country in rivers.items():
    print("The " + river.title() + " flows through " + country.title() + ".")

print("\nThe following rivers are included in this data set:")
for river in rivers.keys():
    print("- " + river.title())

print("\nThe following countries are included in this data set:")
for country in rivers.values():
    print("- " + country.title())
```

Output*:

```
The Mississippi flows through United States.
The Yangtze flows through China.
```

```
The Fraser flows through Canada.
The Nile flows through Egypt.
The Kuskokwim flows through Alaska.

The following rivers are included in this data set:
- Mississippi
- Yangtze
- Fraser
- Nile
- Kuskokwim

The following countries are included in this data set:
- United States
- China
- Canada
- Egypt
- Alaska
```

*Alaska is natuurlijk geen land, maar mensen uit Alaska, zoals de schrijver van dit boek, zien dat vaak wel zo.

## 6-6: Enquête

```
favorite_languages = {
    'jen': 'python',
    'sarah': 'c',
    'edward': 'ruby',
    'phil': 'python',
    }

for name, language in favorite_languages.items():
    print(name.title() + "'s favorite language is " +
        language.title() + ".")

print("\n")

coders = ['phil', 'josh', 'david', 'becca', 'sarah', 'matt', 'danielle']
for coder in coders:
    if coder in favorite_languages.keys():
        print("Thank you for taking the poll, " + coder.title() + "!")
    else:
        print(coder.title() + ", what's your favorite programming language?")
```

Output:

```
Jen's favorite language is Python.
Sarah's favorite language is C.
Phil's favorite language is Python.
Edward's favorite language is Ruby.


Thank you for taking the poll, Phil!
Josh, what's your favorite programming language?
David, what's your favorite programming language?
Becca, what's your favorite programming language?
Thank you for taking the poll, Sarah!
Matt, what's your favorite programming language?
Danielle, what's your favorite programming language?
```

## 6-7: Personen

```
# Make an empty list to store people in.
people = []

# Define some people, and add them to the list.
person = {
    'first_name': 'eric',
```

```
    'last_name': 'matthes',
    'age': 43,
    'city': 'sitka',
    }
people.append(person)

person = {
    'first_name': 'ever',
    'last_name': 'matthes',
    'age': 5,
    'city': 'sitka',
    }
people.append(person)

person = {
    'first_name': 'willie',
    'last_name': 'matthes',
    'age': 8,
    'city': 'sitka',
    }
people.append(person)

# Display all of the information in the dictionary.
for person in people:
    name = person['first_name'].title() + " " + person['last_name'].title()
    age = str(person['age'])
    city = person['city'].title()

    print(name + ", of " + city + ", is " + age + " years old.")
```

Output:

```
Eric Matthes, of Sitka, is 43 years old.
Ever Matthes, of Sitka, is 5 years old.
Willie Matthes, of Sitka, is 8 years old.
```

## 6-8: Huisdieren

Let op: Toen ik besloot oplossingen te plaatsen en complete programma's te schrijven om elke oefening op te lossen, besefte ik dat dit probleem niet zo goed was geformuleerd als het had moeten zijn. Het is niet echt logisch om elk woordenboek een naam te geven voor het huisdier dat het beschrijft; die informatie moet eigenlijk in het woordenboek worden opgenomen, in plaats van dat deze als de naam van het woordenboek wordt gebruikt. Deze oplossing weerspiegelt die aanpak.

```
# Make an empty list to store the pets in.
pets = []

# Make individual pets, and store each one in the list.
pet = {
    'animal type': 'python',
    'name': 'john',
    'owner': 'guido',
    'weight': 43,
    'eats': 'bugs',
}
pets.append(pet)

pet = {
    'animal type': 'chicken',
    'name': 'clarence',
    'owner': 'tiffany',
    'weight': 2,
    'eats': 'seeds',
}
pets.append(pet)

pet = {
```

```
    'animal type': 'dog',
    'name': 'peso',
    'owner': 'eric',
    'weight': 37,
    'eats': 'shoes',
}
pets.append(pet)

# Display information about each pet.
for pet in pets:
    print("\nHere's what I know about " + pet['name'].title() + ":")
    for key, value in pet.items():
        print("\t" + key + ": " + str(value))
```

Output:

```
Here's what I know about John:
    weight: 43
    animal type: python
    name: john
    owner: guido
    eats: bugs

Here's what I know about Clarence:
    weight: 2
    animal type: chicken
    name: clarence
    owner: tiffany
    eats: seeds

Here's what I know about Peso:
    weight: 37
    animal type: dog
    name: peso
    owner: eric
    eats: shoes
```

## 6-9: Favoriete plaatsen

```
favorite_places = {
    'eric': ['bear mountain', 'death valley', 'tierra del fuego'],
    'erin': ['hawaii', 'iceland'],
    'ever': ['mt. verstovia', 'the playground', 'south carolina']
    }

for name, places in favorite_places.items():
    print("\n" + name.title() + " likes the following places:")
    for place in places:
        print("- " + place.title())
```

Output:

```
Ever likes the following places:
- Mt. Verstovia
- The Playground
- South Carolina

Erin likes the following places:
- Hawaii
- Iceland

Eric likes the following places:
- Bear Mountain
- Death Valley
- Tierra Del Fuego
```

## 6-10: Favoriete getallen

```
favorite_numbers = {
    'mandy': [42, 17],
    'micah': [42, 39, 56],
    'gus': [7, 12],
    }

for name, numbers in favorite_numbers.items():
    print("\n" + name.title() + " likes the following numbers:")
    for number in numbers:
        print("  " + str(number))
```

Output:

```
Micah likes the following numbers:
  42
  39
  56

Mandy likes the following numbers:
  42
  17

Gus likes the following numbers:
  7
  12
```

## 6-11: Steden

```
cities = {
    'santiago': {
        'country': 'chile',
        'population': 6158080,
        'nearby mountains': 'andes',
        },
    'talkeetna': {
        'country': 'alaska',
        'population': 876,
        'nearby mountains': 'alaska range',
        },
    'kathmandu': {
        'country': 'nepal',
        'population': 1003285,
        'nearby mountains': 'himilaya',
        }
    }

for city, city_info in cities.items():
    country = city_info['country'].title()
    population = city_info['population']
    mountains = city_info['nearby mountains'].title()

    print("\n" + city.title() + " is in " + country + ".")
    print("  It has a population of about " + str(population) + ".")
    print("  The " + mountains + " mountains are nearby.")
```

Output:

```
Santiago is in Chile.
  It has a population of about 6158080.
  The Andes mountains are nearby.

Kathmandu is in Nepal.
  It has a population of about 1003285.
  The Himilaya mountains are nearby.
```

```
Talkeetna is in Alaska.
  It has a population of about 876.
  The Alaska Range mountains are nearby.
```

**Let op:** Sublime Text voert geen programma's uit die de gebruiker om invoer vragen. U kunt Sublime Text gebruiken om programma's te schrijven die om invoer vragen, maar u moet deze programma's vanaf een terminal uitvoeren. Zie ‘Python-programma's vanuit een terminal uitvoeren’ op pagina 46.

## 7-1: Huurauto

```
car = input("What kind of car would you like? ")

print("Let me see if I can find you a " + car.title() + ".")
```

Output:

```
What kind of car would you like? Toyota Tacoma
Let me see if I can find you a Toyota Tacoma.
```

## 7-2: Een tafeltje in een restaurant

```
party_size = input("How many people are in your dinner party tonight? ")
party_size = int(party_size)

if party_size > 8:
    print("I'm sorry, you'll have to wait for a table.")
else:
    print("Your table is ready.")
```

Output:

```
How many people are in your dinner party tonight? 12
I'm sorry, you'll have to wait for a table.
```

of:

```
How many people are in your dinner party tonight? 6
Your table is ready.
```

## 7-3: Veelvoud van tien

```
number = input("Give me a number, please: ")
number = int(number)

if number % 10 == 0:
    print(str(number) + " is a multiple of 10.")
else:
    print(str(number) + " is not a multiple of 10.")
```

Output:

```
Give me a number, please: 23
23 is not a multiple of 10.
```

of:

```
Give me a number, please: 90
90 is a multiple of 10.
```

## 7-4: Pizza toppings

```
prompt = "\nWhat topping would you like on your pizza?"
prompt += "\nEnter 'quit' when you are finished: "

while True:
    topping = input(prompt)
    if topping != 'quit':
        print("  I'll add " + topping + " to your pizza.")
    else:
        break
```

Output:

```
What topping would you like on your pizza?
Enter 'quit' when you are finished: pepperoni
  I'll add pepperoni to your pizza.

What topping would you like on your pizza?
Enter 'quit' when you are finished: sausage
  I'll add sausage to your pizza.

What topping would you like on your pizza?
Enter 'quit' when you are finished: bacon
  I'll add bacon to your pizza.

What topping would you like on your pizza?
Enter 'quit' when you are finished: quit
```

## 7-5: Filmkaartjes

```
prompt = "How old are you?"
prompt += "\nEnter 'quit' when you are finished. "

while True:
    age = input(prompt)
    if age == 'quit':
        break
    age = int(age)

    if age < 3:
        print("  You get in free!")
    elif age < 13:
        print("  Your ticket is $10.")
    else:
        print("  Your ticket is $15.")
```

Output:

```
How old are you?
Enter 'quit' when you are finished. 2
  You get in free!
How old are you?
Enter 'quit' when you are finished. 3
  Your ticket is $10.
How old are you?
Enter 'quit' when you are finished. 12
  Your ticket is $10.
How old are you?
Enter 'quit' when you are finished. 18
  Your ticket is $15.
How old are you?
Enter 'quit' when you are finished. quit
```

## 7-8: Lunchroom

```
sandwich_orders = ['veggie', 'grilled cheese', 'turkey', 'roast beef']
finished_sandwiches = []

while sandwich_orders:
    current_sandwich = sandwich_orders.pop()
    print("I'm working on your " + current_sandwich + " sandwich.")
    finished_sandwiches.append(current_sandwich)

print("\n")
for sandwich in finished_sandwiches:
    print("I made a " + sandwich + " sandwich.")
```

Output:

```
I'm working on your roast beef sandwich.
I'm working on your turkey sandwich.
I'm working on your grilled cheese sandwich.
I'm working on your veggie sandwich.

I made a roast beef sandwich.
I made a turkey sandwich.
I made a grilled cheese sandwich.
I made a veggie sandwich.
```

## 7-9: Geen pastrami

```
sandwich_orders = [
    'pastrami', 'veggie', 'grilled cheese', 'pastrami',
    'turkey', 'roast beef', 'pastrami']
finished_sandwiches = []

print("I'm sorry, we're all out of pastrami today.")
while 'pastrami' in sandwich_orders:
    sandwich_orders.remove('pastrami')

print("\n")
while sandwich_orders:
    current_sandwich = sandwich_orders.pop()
    print("I'm working on your " + current_sandwich + " sandwich.")
    finished_sandwiches.append(current_sandwich)

print("\n")
for sandwich in finished_sandwiches:
    print("I made a " + sandwich + " sandwich.")
```

Output:

```
I'm sorry, we're all out of pastrami today.

I'm working on your roast beef sandwich.
I'm working on your turkey sandwich.
I'm working on your grilled cheese sandwich.
I'm working on your veggie sandwich.

I made a roast beef sandwich.
I made a turkey sandwich.
I made a grilled cheese sandwich.
I made a veggie sandwich.
```

## 7-10: Droomvakantie

```
name_prompt = "\nWhat's your name? "
place_prompt = "If you could visit one place in the world, where would it be? "
continue_prompt = "\nWould you like to let someone else respond? (yes/no) "
```

```
# Responses will be stored in the form {name: place}.
responses = {}

while True:
    # Ask the user where they'd like to go.
    name = input(name_prompt)
    place = input(place_prompt)

    # Store the response.
    responses[name] = place

    # Ask if there's anyone else responding.
    repeat = input(continue_prompt)
    if repeat != 'yes':
        break

# Show results of the survey.
print("\n--- Results ---")
for name, place in responses.items():
    print(name.title() + " would like to visit " + place.title() + ".")
```

Output:

```
What's your name? eric
If you could visit one place in the world, where would it be? tierra del fuego

Would you like to let someone else respond? (yes/no) yes

What's your name? erin
If you could visit one place in the world, where would it be? iceland

Would you like to let someone else respond? (yes/no) yes

What's your name? ever
If you could visit one place in the world, where would it be? death valley

Would you like to let someone else respond? (yes/no) no

--- Results ---
Ever would like to visit Death Valley.
Erin would like to visit Iceland.
Eric would like to visit Tierra Del Fuego.
```

## 8-1: Bericht

```
def display_message():
    """Display a message about what I'm learning."""
    msg = "I'm learning to store code in functions."
    print(msg)

display_message()
```

Output:

```
I'm learning to store code in functions.
```

## 8-2: Favoriete boek

```
def favorite_book(title):
    """Display a message about someone's favorite book."""
    print(title + " is one of my favorite books.")

favorite_book('The Abstract Wild')
```

Output:

```
The Abstract Wild is one of my favorite books.
```

## 8-3: T-Shirt

```
def make_shirt(size, message):
    """Summarize the shirt that's going to be made."""
    print("\nI'm going to make a " + size + " t-shirt.")
    print('It will say, "' + message + '"')

make_shirt('large', 'I love Python!')
make_shirt(message="Readability counts.", size='medium')
```

Output:

```
I'm going to make a large t-shirt.
It will say, "I love Python!"

I'm going to make a medium t-shirt.
It will say, "Readability counts."
```

## 8-4: Grote shirts

```
def make_shirt(size='large', message='I love Python!'):
    """Summarize the shirt that's going to be made."""
    print("\nI'm going to make a " + size + " t-shirt.")
    print('It will say, "' + message + '"')

make_shirt()
make_shirt(size='medium')
make_shirt('small', 'Programmers are loopy.')
```

Output:

```
I'm going to make a large t-shirt.
It will say, "I love Python!"

I'm going to make a medium t-shirt.
It will say, "I love Python!"

I'm going to make a small t-shirt.
It will say, "Programmers are loopy."
```

## 8-5: Steden

```
def describe_city(city, country='chile'):
    """Describe a city."""
    msg = city.title() + " is in " + country.title() + "."
    print(msg)

describe_city('santiago')
describe_city('reykjavik', 'iceland')
describe_city('punta arenas')
```

Output:

```
Santiago is in Chile.
Reykjavik is in Iceland.
Punta Arenas is in Chile.
```

## 8-6: Namen van steden

```
def city_country(city, country):
    """Return a string like 'Santiago, Chile'."""
    return(city.title() + ", " + country.title())

city = city_country('santiago', 'chile')
print(city)

city = city_country('ushuaia', 'argentina')
print(city)

city = city_country('longyearbyen', 'svalbard')
print(city)
```

Output:

```
Santiago, Chile
Ushuaia, Argentina
Longyearbyen, Svalbard
```

## 8-7: Album

Eenvoudige versie:

```
def make_album(artist, title):
    """Build a dictionary containing information about an album."""
    album_dict = {
        'artist': artist.title(),
        'title': title.title(),
        }
    return album_dict

album = make_album('metallica', 'ride the lightning')
print(album)

album = make_album('beethoven', 'ninth symphony')
print(album)

album = make_album('willie nelson', 'red-headed stranger')
print(album)
```

Output:

```
{'title': 'Ride The Lightning', 'artist': 'Metallica'}
{'title': 'Ninth Symphony', 'artist': 'Beethoven'}
{'title': 'Red-Headed Stranger', 'artist': 'Willie Nelson'}
```

Met tracks (nummers):

```
def make_album(artist, title, tracks=0):
    """Build a dictionary containing information about an album."""
    album_dict = {
        'artist': artist.title(),
        'title': title.title(),
        }
    if tracks:
        album_dict['tracks'] = tracks
    return album_dict

album = make_album('metallica', 'ride the lightning')
print(album)

album = make_album('beethoven', 'ninth symphony')
print(album)
```

```
album = make_album('willie nelson', 'red-headed stranger')
print(album)

album = make_album('iron maiden', 'piece of mind', tracks=8)
print(album)
```

Output:

```
{'artist': 'Metallica', 'title': 'Ride The Lightning'}
{'artist': 'Beethoven', 'title': 'Ninth Symphony'}
{'artist': 'Willie Nelson', 'title': 'Red-Headed Stranger'}
{'tracks': 8, 'artist': 'Iron Maiden', 'title': 'Piece Of Mind'}
```

## 8-8: Albums van gebruikers

```
def make_album(artist, title, tracks=0):
    """Build a dictionary containing information about an album."""
    album_dict = {
        'artist': artist.title(),
        'title': title.title(),
        }
    if tracks:
        album_dict['tracks'] = tracks
    return album_dict

# Prepare the prompts.
title_prompt = "\nWhat album are you thinking of? "
artist_prompt = "Who's the artist? "

# Let the user know how to quit.
print("Enter 'quit' at any time to stop.")

while True:
    title = input(title_prompt)
    if title == 'quit':
        break

    artist = input(artist_prompt)
    if artist == 'quit':
        break

    album = make_album(artist, title)
    print(album)

print("\nThanks for responding!")
```

Output:

```
Enter 'quit' at any time to stop.

What album are you thinking of? number of the beast
Who's the artist? iron maiden
{'artist': 'Iron Maiden', 'title': 'Number Of The Beast'}

What album are you thinking of? touch of class
Who's the artist? angel romero
{'artist': 'Angel Romero', 'title': 'Touch Of Class'}

What album are you thinking of? rust in peace
Who's the artist? megadeth
{'artist': 'Megadeth', 'title': 'Rust In Peace'}

What album are you thinking of? quit

Thanks for responding!
```

## 8-9: Goochelaars

```
def show_magicians(magicians):
    """Print the name of each magician in the list."""
    for magician in magicians:
        print(magician.title())

magicians = ['harry houdini', 'david blaine', 'teller']
show_magicians(magicians)
```

Output:

```
Harry Houdini
David Blaine
Teller
```

## 8-10: Grote goochelaars

```
def show_magicians(magicians):
    """Print the name of each magician in the list."""
    for magician in magicians:
        print(magician)

def make_great(magicians):
    """Add 'the Great!' to each magician's name."""
    # Build a new list to hold the great musicians.
    great_magicians = []

    # Make each magician great, and add it to great_magicians.
    while magicians:
        magician = magicians.pop()
        great_magician = magician + ' the Great'
        great_magicians.append(great_magician)

    # Add the great magicians back into magicians.
    for great_magician in great_magicians:
        magicians.append(great_magician)

magicians = ['Harry Houdini', 'David Blaine', 'Teller']
show_magicians(magicians)

print("\n")
make_great(magicians)
show_magicians(magicians)
```

Output:

```
Harry Houdini
David Blaine
Teller

Teller the Great
David Blaine the Great
Harry Houdini the Great
```

## 8-11: Ongewijzigde goochelaars

```
def show_magicians(magicians):
    """Print the name of each magician in the list."""
    for magician in magicians:
        print(magician)

def make_great(magicians):
    """Add 'the Great!' to each magician's name."""
    # Build a new list to hold the great musicians.
    great_magicians = []
```

```
    # Make each magician great, and add it to great_magicians.
    while magicians:
        magician = magicians.pop()
        great_magician = magician + ' the Great'
        great_magicians.append(great_magician)

    # Add the great magicians back into magicians.
    for great_magician in great_magicians:
        magicians.append(great_magician)

    return magicians

magicians = ['Harry Houdini', 'David Blaine', 'Teller']
show_magicians(magicians)

print("\nGreat magicians:")
great_magicians = make_great(magicians[:])
show_magicians(great_magicians)

print("\nOriginal magicians:")
show_magicians(magicians)
```

Output:

```
Harry Houdini
David Blaine
Teller

Great magicians:
Teller the Great
David Blaine the Great
Harry Houdini the Great

Original magicians:
Harry Houdini
David Blaine
Teller
```

## 8-12: Sandwiches

```
def make_sandwich(*items):
    """Make a sandwich with the given items."""
    print("\nI'll make you a great sandwich:")
    for item in items:
        print("  ...adding " + item + " to your sandwich.")
    print("Your sandwich is ready!")

make_sandwich('roast beef', 'cheddar cheese', 'lettuce', 'honey dijon')
make_sandwich('turkey', 'apple slices', 'honey mustard')
make_sandwich('peanut butter', 'strawberry jam')
```

Output:

```
I'll make you a great sandwich:
  ...adding roast beef to your sandwich.
  ...adding cheddar cheese to your sandwich.
  ...adding lettuce to your sandwich.
  ...adding honey dijon to your sandwich.
Your sandwich is ready!

I'll make you a great sandwich:
  ...adding turkey to your sandwich.
  ...adding apple slices to your sandwich.
  ...adding honey mustard to your sandwich.
Your sandwich is ready!

I'll make you a great sandwich:
  ...adding peanut butter to your sandwich.
```

```
  ...adding strawberry jam to your sandwich.
Your sandwich is ready!
```

## 8-14: Auto's

```
def make_car(manufacturer, model, **options):
    """Make a dictionary representing a car."""
    car_dict = {
        'manufacturer': manufacturer.title(),
        'model': model.title(),
        }
    for option, value in options.items():
        car_dict[option] = value

    return car_dict

my_outback = make_car('subaru', 'outback', color='blue', tow_package=True)
print(my_outback)

my_accord = make_car('honda', 'accord', year=1991, color='white',
        headlights='popup')
print(my_accord)
```

Output:

```
{'manufacturer': 'Subaru', 'color': 'blue', 'tow_package': True, 'model': 'Outback'}
{'year': 1991, 'manufacturer': 'Honda', 'color': 'white', 'headlights': 'popup', 'model':
'Accord'}
```

## 8-15: Modellen afdrukken

printing_functions.py:

```
"""Functions related to printing 3d models."""

def print_models(unprinted_designs, completed_models):
    """
    Simulate printing each design, until there are none left.
    Move each design to completed_models after printing.
    """
    while unprinted_designs:
        current_design = unprinted_designs.pop()

        # Simulate creating a 3d print from the design.
        print("Printing model: " + current_design)
        completed_models.append(current_design)

def show_completed_models(completed_models):
    """Show all the models that were printed."""
    print("\nThe following models have been printed:")
    for completed_model in completed_models:
        print(completed_model)
```

printing_models.py:

```
import printing_functions as pf

unprinted_designs = ['iphone case', 'robot pendant', 'dodecahedron']
completed_models = []

pf.print_models(unprinted_designs, completed_models)
pf.show_completed_models(completed_models)
```

Output:

```
Printing model: dodecahedron
```

```
Printing model: robot pendant
Printing model: iphone case

The following models have been printed:
dodecahedron
robot pendant
iphone case
```

## 9-1: Restaurant

```
class Restaurant():
    """A class representing a restaurant."""

    def __init__(self, name, cuisine_type):
        """Initialize the restaurant."""
        self.name = name.title()
        self.cuisine_type = cuisine_type

    def describe_restaurant(self):
        """Display a summary of the restaurant."""
        msg = self.name + " serves wonderful " + self.cuisine_type + "."
        print("\n" + msg)

    def open_restaurant(self):
        """Display a message that the restaurant is open."""
        msg = self.name + " is open. Come on in!"
        print("\n" + msg)

restaurant = Restaurant('the mean queen', 'pizza')
print(restaurant.name)
print(restaurant.cuisine_type)

restaurant.describe_restaurant()
restaurant.open_restaurant()
```

Output:

```
The Mean Queen
pizza

The Mean Queen serves wonderful pizza.

The Mean Queen is open. Come on in!
```

## 9-2: Drie restaurants

```
class Restaurant():
    """A class representing a restaurant."""

    def __init__(self, name, cuisine_type):
        """Initialize the restaurant."""
        self.name = name.title()
        self.cuisine_type = cuisine_type

    def describe_restaurant(self):
        """Display a summary of the restaurant."""
        msg = self.name + " serves wonderful " + self.cuisine_type + "."
        print("\n" + msg)

    def open_restaurant(self):
        """Display a message that the restaurant is open."""
        msg = self.name + " is open. Come on in!"
        print("\n" + msg)

mean_queen = Restaurant('the mean queen', 'pizza')
mean_queen.describe_restaurant()

ludvigs = Restaurant("ludvig's bistro", 'seafood')
ludvigs.describe_restaurant()
```

```
mango_thai = Restaurant('mango thai', 'thai food')
mango_thai.describe_restaurant()
```

Output:

```
The Mean Queen serves wonderful pizza.

Ludvig'S Bistro serves wonderful seafood.

Mango Thai serves wonderful thai food.
```

## 9-3: Gebruikers

```
class User():
    """Represent a simple user profile."""

    def __init__(self, first_name, last_name, username, email, location):
        """Initialize the user."""
        self.first_name = first_name.title()
        self.last_name = last_name.title()
        self.username = username
        self.email = email
        self.location = location.title()

    def describe_user(self):
        """Display a summary of the user's information."""
        print("\n" + self.first_name + " " + self.last_name)
        print("  Username: " + self.username)
        print("  Email: " + self.email)
        print("  Location: " + self.location)

    def greet_user(self):
        """Display a personalized greeting to the user."""
        print("\nWelcome back, " + self.username + "!")

eric = User('eric', 'matthes', 'e_matthes', 'e_matthes@example.com', 'alaska')
eric.describe_user()
eric.greet_user()

willie = User('willie', 'burger', 'willieburger', 'wb@example.com', 'alaska')
willie.describe_user()
willie.greet_user()
```

Output:

```
Eric Matthes
  Username: e_matthes
  Email: e_matthes@example.com
  Location: Alaska

Welcome back, e_matthes!

Willie Burger
  Username: willieburger
  Email: wb@example.com
  Location: Alaska

Welcome back, willieburger!
```

## 9-4: Aantal bediende gasten

```
class Restaurant():
    """A class representing a restaurant."""

    def __init__(self, name, cuisine_type):
        """Initialize the restaurant."""
        self.name = name.title()
```

```
        self.cuisine_type = cuisine_type
        self.number_served = 0

    def describe_restaurant(self):
        """Display a summary of the restaurant."""
        msg = self.name + " serves wonderful " + self.cuisine_type + "."
        print("\n" + msg)

    def open_restaurant(self):
        """Display a message that the restaurant is open."""
        msg = self.name + " is open. Come on in!"
        print("\n" + msg)

    def set_number_served(self, number_served):
        """Allow user to set the number of customers that have been served."""
        self.number_served = number_served

    def increment_number_served(self, additional_served):
        """Allow user to increment the number of customers served."""
        self.number_served += additional_served


restaurant = Restaurant('the mean queen', 'pizza')
restaurant.describe_restaurant()

print("\nNumber served: " + str(restaurant.number_served))
restaurant.number_served = 430
print("Number served: " + str(restaurant.number_served))

restaurant.set_number_served(1257)
print("Number served: " + str(restaurant.number_served))

restaurant.increment_number_served(239)
print("Number served: " + str(restaurant.number_served))
```

Output:

```
The Mean Queen serves wonderful pizza.

Number served: 0
Number served: 430
Number served: 1257
Number served: 1496
```

## 9-5: Inlogpogingen

```
class User():
    """Represent a simple user profile."""

    def __init__(self, first_name, last_name, username, email, location):
        """Initialize the user."""
        self.first_name = first_name.title()
        self.last_name = last_name.title()
        self.username = username
        self.email = email
        self.location = location.title()
        self.login_attempts = 0

    def describe_user(self):
        """Display a summary of the user's information."""
        print("\n" + self.first_name + " " + self.last_name)
        print("  Username: " + self.username)
        print("  Email: " + self.email)
        print("  Location: " + self.location)

    def greet_user(self):
        """Display a personalized greeting to the user."""
        print("\nWelcome back, " + self.username + "!")
```

```
    def increment_login_attempts(self):
        """Increment the value of login_attempts."""
        self.login_attempts += 1

    def reset_login_attempts(self):
        """Reset login_attempts to 0."""
        self.login_attempts = 0

eric = User('eric', 'matthes', 'e_matthes', 'e_matthes@example.com', 'alaska')
eric.describe_user()
eric.greet_user()

print("\nMaking 3 login attempts...")
eric.increment_login_attempts()
eric.increment_login_attempts()
eric.increment_login_attempts()
print("  Login attempts: " + str(eric.login_attempts))

print("Resetting login attempts...")
eric.reset_login_attempts()
print("  Login attempts: " + str(eric.login_attempts))
```

Output:

```
Eric Matthes
  Username: e_matthes
  Email: e_matthes@example.com
  Location: Alaska

Welcome back, e_matthes!

Making 3 login attempts...
  Login attempts: 3
Resetting login attempts...
  Login attempts: 0
```

## 9-6: De ijscoman

```
class Restaurant():
    """A class representing a restaurant."""

    def __init__(self, name, cuisine_type):
        """Initialize the restaurant."""
        self.name = name.title()
        self.cuisine_type = cuisine_type
        self.number_served = 0

    def describe_restaurant(self):
        """Display a summary of the restaurant."""
        msg = self.name + " serves wonderful " + self.cuisine_type + "."
        print("\n" + msg)

    def open_restaurant(self):
        """Display a message that the restaurant is open."""
        msg = self.name + " is open. Come on in!"
        print("\n" + msg)

    def set_number_served(self, number_served):
        """Allow user to set the number of customers that have been served."""
        self.number_served = number_served

    def increment_number_served(self, additional_served):
        """Allow user to increment the number of customers served."""
        self.number_served += additional_served


class IceCreamStand(Restaurant):
    """Represent an ice cream stand."""
```

```
    def __init__(self, name, cuisine_type='ice_cream'):
        """Initialize an ice cream stand."""
        super().__init__(name, cuisine_type)
        self.flavors = []

    def show_flavors(self):
        """Display the flavors available."""
        print("\nWe have the following flavors available:")
        for flavor in self.flavors:
            print("- " + flavor.title())


big_one = IceCreamStand('The Big One')
big_one.flavors = ['vanilla', 'chocolate', 'black cherry']

big_one.describe_restaurant()
big_one.show_flavors()
```

Output:

```
The Big One serves wonderful ice_cream.

We have the following flavors available:
- Vanilla
- Chocolate
- Black Cherry
```

## 9-7: Beheerder

```
class User():
    """Represent a simple user profile."""

    def __init__(self, first_name, last_name, username, email, location):
        """Initialize the user."""
        self.first_name = first_name.title()
        self.last_name = last_name.title()
        self.username = username
        self.email = email
        self.location = location.title()
        self.login_attempts = 0

    def describe_user(self):
        """Display a summary of the user's information."""
        print("\n" + self.first_name + " " + self.last_name)
        print("  Username: " + self.username)
        print("  Email: " + self.email)
        print("  Location: " + self.location)

    def greet_user(self):
        """Display a personalized greeting to the user."""
        print("\nWelcome back, " + self.username + "!")

    def increment_login_attempts(self):
        """Increment the value of login_attempts."""
        self.login_attempts += 1

    def reset_login_attempts(self):
        """Reset login_attempts to 0."""
        self.login_attempts = 0


class Admin(User):
    """A user with administrative privileges."""

    def __init__(self, first_name, last_name, username, email, location):
        """Initialize the admin."""
        super().__init__(first_name, last_name, username, email, location)
        self.privileges = []
```

```
    def show_privileges(self):
        """Display the privileges this administrator has."""
        print("\nPrivileges:")
        for privilege in self.privileges:
            print("- " + privilege)


eric = Admin('eric', 'matthes', 'e_matthes', 'e_matthes@example.com', 'alaska')
eric.describe_user()

eric.privileges = [
    'can reset passwords',
    'can moderate discussions',
    'can suspend accounts',
    ]

eric.show_privileges()
```

Output:

```
Eric Matthes
  Username: e_matthes
  Email: e_matthes@example.com
  Location: Alaska

Privileges:
- can reset passwords
- can moderate discussions
- can suspend accounts
```

## 9-8: Privileges

```
class User():
    """Represent a simple user profile."""

    def __init__(self, first_name, last_name, username, email, location):
        """Initialize the user."""
        self.first_name = first_name.title()
        self.last_name = last_name.title()
        self.username = username
        self.email = email
        self.location = location.title()
        self.login_attempts = 0

    def describe_user(self):
        """Display a summary of the user's information."""
        print("\n" + self.first_name + " " + self.last_name)
        print("  Username: " + self.username)
        print("  Email: " + self.email)
        print("  Location: " + self.location)

    def greet_user(self):
        """Display a personalized greeting to the user."""
        print("\nWelcome back, " + self.username + "!")

    def increment_login_attempts(self):
        """Increment the value of login_attempts."""
        self.login_attempts += 1

    def reset_login_attempts(self):
        """Reset login_attempts to 0."""
        self.login_attempts = 0


class Admin(User):
    """A user with administrative privileges."""

    def __init__(self, first_name, last_name, username, email, location):
        """Initialize the admin."""
```

```
        super().__init__(first_name, last_name, username, email, location)

        # Initialize an empty set of privileges.
        self.privileges = Privileges()

class Privileges():
    """A class to store an admin's privileges."""

    def __init__(self, privileges=[]):
        self.privileges = privileges

    def show_privileges(self):
        print("\nPrivileges:")
        if self.privileges:
            for privilege in self.privileges:
                print("- " + privilege)
        else:
            print("- This user has no privileges.")


eric = Admin('eric', 'matthes', 'e_matthes', 'e_matthes@example.com', 'alaska')
eric.describe_user()

eric.privileges.show_privileges()

print("\nAdding privileges...")
eric_privileges = [
    'can reset passwords',
    'can moderate discussions',
    'can suspend accounts',
    ]
eric.privileges.privileges = eric_privileges
eric.privileges.show_privileges()
```

Output:

```
Eric Matthes
  Username: e_matthes
  Email: e_matthes@example.com
  Location: Alaska

Privileges:
- This user has no privileges.

Adding privileges...

Privileges:
- can reset passwords
- can moderate discussions
- can suspend accounts
```

## 9-9: Accu-upgrade

```
class Car():
    """A simple attempt to represent a car."""

    def __init__(self, manufacturer, model, year):
        """Initialize attributes to describe a car."""
        self.manufacturer = manufacturer
        self.model = model
        self.year = year
        self.odometer_reading = 0

    def get_descriptive_name(self):
        """Return a neatly formatted descriptive name."""
        long_name = str(self.year) + ' ' + self.manufacturer + ' ' + self.model
        return long_name.title()

    def read_odometer(self):
```

```
        """Print a statement showing the car's mileage."""
        print("This car has " + str(self.odometer_reading) + " miles on it.")

    def update_odometer(self, mileage):
        """
        Set the odometer reading to the given value.
        Reject the change if it attempts to roll the odometer back.
        """
        if mileage >= self.odometer_reading:
            self.odometer_reading = mileage
        else:
            print("You can't roll back an odometer!")

    def increment_odometer(self, miles):
        """Add the given amount to the odometer reading."""
        self.odometer_reading += miles

class Battery():
    """A simple attempt to model a battery for an electric car."""

    def __init__(self, battery_size=60):
        """Initialize the batteery's attributes."""
        self.battery_size = battery_size

    def describe_battery(self):
        """Print a statement describing the battery size."""
        print("This car has a " + str(self.battery_size) + "-kWh battery.")


    def get_range(self):
        """Print a statement about the range this battery provides."""
        if self.battery_size == 60:
            range = 140
        elif self.battery_size == 85:
            range = 185

        message = "This car can go approximately " + str(range)
        message += " miles on a full charge."
        print(message)

    def upgrade_battery(self):
        """Upgrade the battery if possible."""
        if self.battery_size == 60:
            self.battery_size = 85
            print("Upgraded the battery to 85 kWh.")
        else:
            print("The battery is already upgraded.")


class ElectricCar(Car):
    """Models aspects of a car, specific to electric vehicles."""

    def __init__(self, manufacturer, model, year):
        """
        Initialize attributes of the parent class.
        Then initialize attributes specific to an electric car.
        """
        super().__init__(manufacturer, model, year)
        self.battery = Battery()


print("Make an electric car, and check the battery:")
my_tesla = ElectricCar('tesla', 'model s', 2016)
my_tesla.battery.describe_battery()

print("\nUpgrade the battery, and check it again:")
my_tesla.battery.upgrade_battery()
my_tesla.battery.describe_battery()

print("\nTry upgrading the battery a second time.")
my_tesla.battery.upgrade_battery()
```

```
my_tesla.battery.describe_battery()
```

Output:

```
Make an electric car, and check the battery:
This car has a 60-kWh battery.

Upgrade the battery, and check it again:
Upgraded the battery to 85 kWh.
This car has a 85-kWh battery.

Try upgrading the battery a second time.
The battery is already upgraded.
This car has a 85-kWh battery.
```

## 9-10: Geïmporteerd restaurant

restaurant.py:

```
"""A class representing a restaurant."""

class Restaurant():
    """A class representing a restaurant."""

    def __init__(self, name, cuisine_type):
        """Initialize the restaurant."""
        self.name = name.title()
        self.cuisine_type = cuisine_type
        self.number_served = 0

    def describe_restaurant(self):
        """Display a summary of the restaurant."""
        msg = self.name + " serves wonderful " + self.cuisine_type + "."
        print("\n" + msg)

    def open_restaurant(self):
        """Display a message that the restaurant is open."""
        msg = self.name + " is open. Come on in!"
        print("\n" + msg)

    def set_number_served(self, number_served):
        """Allow user to set the number of customers that have been served."""
        self.number_served = number_served

    def increment_number_served(self, additional_served):
        """Allow user to increment the number of customers served."""
        self.number_served += additional_served
```

my_restaurant.py:

```
from restaurant import Restaurant

channel_club = Restaurant('the channel club', 'steak and seafood')
channel_club.describe_restaurant()
channel_club.open_restaurant()
```

Output:

```
The Channel Club serves wonderful steak and seafood.

The Channel Club is open. Come on in!
```

## 9-11: Geïmporteerde beheerder

user.py:

```
"""A collection of classes for modeling users."""

class User():
    """Represent a simple user profile."""

    def __init__(self, first_name, last_name, username, email, location):
        """Initialize the user."""
        self.first_name = first_name.title()
        self.last_name = last_name.title()
        self.username = username
        self.email = email
        self.location = location.title()
        self.login_attempts = 0

    def describe_user(self):
        """Display a summary of the user's information."""
        print("\n" + self.first_name + " " + self.last_name)
        print("  Username: " + self.username)
        print("  Email: " + self.email)
        print("  Location: " + self.location)

    def greet_user(self):
        """Display a personalized greeting to the user."""
        print("\nWelcome back, " + self.username + "!")

    def increment_login_attempts(self):
        """Increment the value of login_attempts."""
        self.login_attempts += 1

    def reset_login_attempts(self):
        """Reset login_attempts to 0."""
        self.login_attempts = 0


class Admin(User):
    """A user with administrative privileges."""

    def __init__(self, first_name, last_name, username, email, location):
        """Initialize the admin."""
        super().__init__(first_name, last_name, username, email, location)

        # Initialize an empty set of privileges.
        self.privileges = Privileges([])


class Privileges():
    """Stores privileges associated with an Admin account."""

    def __init__(self, privileges):
        """Initialize the privileges object."""
        self.privilege = privileges

    def show_privileges(self):
        """Display the privileges this administrator has."""
        for privilege in self.privileges:
            print("- " + privilege)
```

my_user.py:

```
from user import Admin

eric = Admin('eric', 'matthes', 'e_matthes', 'e_matthes@example.com', 'alaska')
eric.describe_user()

eric_privileges = [
```

```
    'can reset passwords',
    'can moderate discussions',
    'can suspend accounts',
    ]
eric.privileges.privileges = eric_privileges

print("\nThe admin " + eric.username + " has these privileges: ")
eric.privileges.show_privileges()
```

Output:

```
Eric Matthes
  Username: e_matthes
  Email: e_matthes@example.com
  Location: Alaska

The admin e_matthes has these privileges:
- can reset passwords
- can moderate discussions
- can suspend accounts
```

## 9-12: Meerdere modules

user.py:

```
"""A class for modeling users."""

class User():
    """Represent a simple user profile."""

    def __init__(self, first_name, last_name, username, email, location):
        """Initialize the user."""
        self.first_name = first_name.title()
        self.last_name = last_name.title()
        self.username = username
        self.email = email
        self.location = location.title()
        self.login_attempts = 0

    def describe_user(self):
        """Display a summary of the user's information."""
        print("\n" + self.first_name + " " + self.last_name)
        print("  Username: " + self.username)
        print("  Email: " + self.email)
        print("  Location: " + self.location)

    def greet_user(self):
        """Display a personalized greeting to the user."""
        print("\nWelcome back, " + self.username + "!")

    def increment_login_attempts(self):
        """Increment the value of login_attempts."""
        self.login_attempts += 1

    def reset_login_attempts(self):
        """Reset login_attempts to 0."""
        self.login_attempts = 0
```

admin.py:

```
"""A collection of classes for modeling an admin user account."""

from user import User


class Admin(User):
    """A user with administrative privileges."""
```

```
    def __init__(self, first_name, last_name, username, email, location):
        """Initialize the admin."""
        super().__init__(first_name, last_name, username, email, location)

        # Initialize an empty set of privileges.
        self.privileges = Privileges([])


class Privileges():
    """Stores privileges associated with an Admin account."""

    def __init__(self, privileges):
        """Initialize the privileges object."""
        self.privilege = privileges

    def show_privileges(self):
        """Display the privileges this administrator has."""
        for privilege in self.privileges:
            print("- " + privilege)
```

my_admin.py

```
from admin import Admin

eric = Admin('eric', 'matthes', 'e_matthes', 'e_matthes@example.com', 'alaska')
eric.describe_user()

eric_privileges = [
    'can reset passwords',
    'can moderate discussions',
    'can suspend accounts',
    ]
eric.privileges.privileges = eric_privileges

print("\nThe admin " + eric.username + " has these privileges: ")
eric.privileges.show_privileges()
```

Output:

```
Eric Matthes
  Username: e_matthes
  Email: e_matthes@example.com
  Location: Alaska

The admin e_matthes has these privileges:
- can reset passwords
- can moderate discussions
- can suspend accounts
```

## 9-13: OrderedDict herschrijven

**Let op:** In Python 3.6 slaan woordenboeken sleutels op in de juiste volgorde. Dit werkt echter niet gegarandeerd in alle versies van Python, dus u moet nog steeds een OrderedDict gebruiken wanneer u sleutel-waarde-paren nodig hebt die in een bepaalde volgorde moeten worden opgeslagen.

```
from collections import OrderedDict

glossary = OrderedDict()

glossary['string'] = 'A series of characters.'
glossary['comment'] = 'A note in a program that the Python interpreter ignores.'
glossary['list'] = 'A collection of items in a particular order.'
glossary['loop'] = 'Work through a collection of items, one at a time.'
glossary['dictionary'] = "A collection of key-value pairs."
glossary['key'] = 'The first item in a key-value pair in a dictionary.'
glossary['value'] = 'An item associated with a key in a dictionary.'
```

```
glossary['conditional test'] = 'A comparison between two values.'
glossary['float'] = 'A numerical value with a decimal component.'
glossary['boolean expression'] = 'An expression that evaluates to True or False.'

for word, definition in glossary.items():
    print("\n" + word.title() + ": " + definition)
```

Output:

```
String: A series of characters.

Comment: A note in a program that the Python interpreter ignores.

List: A collection of items in a particular order.

Loop: Work through a collection of items, one at a time.

Dictionary: A collection of key-value pairs.

Key: The first item in a key-value pair in a dictionary.

Value: An item associated with a key in a dictionary.

Conditional Test: A comparison between two values.

Float: A numerical value with a decimal component.

Boolean Expression: An expression that evaluates to True or False.
```

## 9-14: Dobbelsteen

```
from random import randint

class Die():
    """Represent a die, which can be rolled."""

    def __init__(self, sides=6):
        """Initialize the die."""
        self.sides = sides

    def roll_die(self):
        """Return a number between 1 and the number of sides."""
        return randint(1, self.sides)

# Make a 6-sided die, and show the results of 10 rolls.
d6 = Die()

results = []
for roll_num in range(10):
    result = d6.roll_die()
    results.append(result)
print("10 rolls of a 6-sided die:")
print(results)

# Make a 10-sided die, and show the results of 10 rolls.
d10 = Die(sides=10)

results = []
for roll_num in range(10):
    result = d10.roll_die()
    results.append(result)
print("\n10 rolls of a 10-sided die:")
print(results)

# Make a 20-sided die, and show the results of 10 rolls.
d20 = Die(sides=20)

results = []
for roll_num in range(10):
```

```
    result = d20.roll_die()
    results.append(result)
print("\n10 rolls of a 20-sided die:")
print(results)
```

Output:

```
10 rolls of a 6-sided die:
[5, 5, 6, 3, 6, 4, 2, 2, 1, 1]

10 rolls of a 10-sided die:
[8, 9, 8, 10, 7, 1, 3, 5, 3, 4]

10 rolls of a 20-sided die:
[4, 3, 18, 17, 3, 1, 13, 12, 5, 14]
```

## 10-1: Python leren

learning_python.txt:

```
In Python you can store as much information as you want.
In Python you can connect pieces of information.
In Python you can model real-world situations.
```

learning_python.py:

```
filename = 'learning_python.txt'

print("--- Reading in the entire file:")
with open(filename) as f:
    contents = f.read()
print(contents)

print("\n--- Looping over the lines:")
with open(filename) as f:
    for line in f:
        print(line.rstrip())

print("\n--- Storing the lines in a list:")
with open(filename) as f:
    lines = f.readlines()

for line in lines:
    print(line.rstrip())
```

Output:

```
--- Reading in the entire file:
In Python you can store as much information as you want.
In Python you can connect pieces of information.
In Python you can model real-world situations.

--- Looping over the lines:
In Python you can store as much information as you want.
In Python you can connect pieces of information.
In Python you can model real-world situations.

--- Storing the lines in a list:
In Python you can store as much information as you want.
In Python you can connect pieces of information.
In Python you can model real-world situations.
```

## 10-2: C leren

```
filename = 'learning_python.txt'

with open(filename) as f:
    lines = f.readlines()

for line in lines:
    # Get rid of newline, then replace Python with C.
    line = line.rstrip()
    print(line.replace('Python', 'C'))
```

Output:

```
In C you can store as much information as you want.
In C you can connect pieces of information.
In C you can model real-world situations.
```

U kunt `rstrip()` en `replace()` op dezelfde regel gebruiken. Dit worden ketenmethoden genoemd. In de volgende code wordt de nieuwe regel verwijderd van het einde van de regel en wordt *Python* vervangen door *C*. De uitvoer is identiek aan de hierboven getoonde code.

```
filename = 'learning_python.txt'

with open(filename) as f:
    lines = f.readlines()

for line in lines:
    # Get rid of newline, then replace Python with C.
    print(line.rstrip().replace('Python', 'C'))
```

## 10-3: Gast

```
name = input("What's your name? ")

filename = 'guest.txt'

with open(filename, 'w') as f:
    f.write(name)
```

Output:

```
What's your name? eric
```

guest.txt:

```
eric
```

## 10-4: Gastenboek

```
filename = 'guest_book.txt'

print("Enter 'quit' when you are finished.")
while True:
    name = input("\nWhat's your name? ")
    if name == 'quit':
        break
    else:
        with open(filename, 'a') as f:
            f.write(name + "\n")
        print("Hi " + name + ", you've been added to the guest book.")
```

Output:

```
Enter 'quit' when you are finished.

What's your name? eric
Hi eric, you've been added to the guest book.

What's your name? willie
Hi willie, you've been added to the guest book.

What's your name? ever
Hi ever, you've been added to the guest book.

What's your name? erin
Hi erin, you've been added to the guest book.

What's your name? quit
```

guest_book.txt:

```
eric
willie
ever
erin
```

## 10-5: Enquête over programmeren

```
filename = 'programming_poll.txt'

responses = []
while True:
    response = input("\nWhy do you like programming? ")
    responses.append(response)

    continue_poll = input("Would you like to let someone else respond? (y/n) ")
    if continue_poll != 'y':
        break

with open(filename, 'a') as f:
    for response in responses:
        f.write(response + "\n")
```

Output:

```
Why do you like programming? Programmers can build almost anything they can imagine.
Would you like to let someone else respond? (y/n) y

Why do you like programming? It's really fun, and really satisfying.
Would you like to let someone else respond? (y/n) y

Why do you like programming? It just never gets old.
Would you like to let someone else respond? (y/n) n
```

programming_poll.txt:

```
Programmers can build almost anything they can imagine.
It's really fun, and really satisfying.
It just never gets old.
```

## 10-6: Optellen

```
try:
    x = input("Give me a number: ")
    x = int(x)

    y = input("Give me another number: ")
```

```
    y = int(y)

except ValueError:
    print("Sorry, I really needed a number.")

else:
    sum = x + y
    print("The sum of " + str(x) + " and " + str(y) + " is " + str(sum) + ".")
```

Output met twee integers:

```
Give me a number: 23
Give me another number: 47
The sum of 23 and 47 is 70.
```

Output zonder numerieke input:

```
Give me a number: 23
Give me another number: fred
Sorry, I really needed a number.
```

## 10-7: Optelcalculator

```
print("Enter 'q' at any time to quit.\n")

while True:
    try:
        x = input("\nGive me a number: ")
        if x == 'q':
            break

        x = int(x)

        y = input("Give me another number: ")
        if y == 'q':
            break

        y = int(y)

    except ValueError:
        print("Sorry, I really needed a number.")

    else:
        sum = x + y
        print("The sum of " + str(x) + " and " + str(y) + " is " + str(sum) + ".")
```

Output:

```
Enter 'q' at any time to quit.

Give me a number: 23
Give me another number: 47
The sum of 23 and 47 is 70.

Give me a number: three
Sorry, I really needed a number.

Give me a number: 3
Give me another number: five
Sorry, I really needed a number.

Give me a number: -12
Give me another number: 20
The sum of -12 and 20 is 8.

Give me a number: q
```

## 10-8: Cats and Dogs

cats.txt:

```
henry
clarence
mildred
```

dogs.txt:

```
willie
annahootz
summit
```

cats_and_dogs.py:

```
filenames = ['cats.txt', 'dogs.txt']

for filename in filenames:
    print("\nReading file: " + filename)
    try:
        with open(filename) as f:
            contents = f.read()
            print(contents)
    except FileNotFoundError:
        print("  Sorry, I can't find that file.")
```

Output met beide bestanden:

```
Reading file: cats.txt
henry
clarence
mildred

Reading file: dogs.txt
willie
annahootz
summit
```

Output na het verplaatsen van *cats.txt:*

```
Reading file: cats.txt
  Sorry, I can't find that file.

Reading file: dogs.txt
willie
annahootz
summit
```

## 10-9: Cats and Dogs (in stilte)

```
filenames = ['cats.txt', 'dogs.txt']

for filename in filenames:

    try:
        with open(filename) as f:
            contents = f.read()

    except FileNotFoundError:
        pass

    else:
        print("\nReading file: " + filename)
        print(contents)
```

Output wanneer beide bestanden er zijn:

```
Reading file: cats.txt
henry
clarence
mildred

Reading file: dogs.txt
willie
annahootz
summit
```

Output wanneer *cats.txt* ontbreekt of is verplaatst:

```
Reading file: dogs.txt
willie
annahootz
summit
```

## 10-11: Favoriete getal

favorite_number_write.py:

```
import json

number = input("What's your favorite number? ")

with open('favorite_number.json', 'w') as f:
    json.dump(number, f)
    print("Thanks! I'll remember that.")
```

Output:

```
What's your favorite number? 42
Thanks! I'll remember that.
```

favorite_number_read.py:

```
import json

with open('favorite_number.json') as f:
    number = json.load(f)

print("I know your favorite number! It's " + str(number) + ".")
```

Output:

```
I know your favorite number! It's 42.
```

## 10-12: Favoriete getal onthouden

```
import json

try:
    with open('favorite_number.json') as f:
        number = json.load(f)
except FileNotFoundError:
    number = input("What's your favorite number? ")
    with open('favorite_number.json', 'w') as f:
        json.dump(number, f)
    print("Thanks, I'll remember that.")
else:
    print("I know your favorite number! It's " + str(number) + ".")
```

Output, eerste uitvoer:

```
What's your favorite number? 42
Thanks, I'll remember that.
```

Output, tweede uitvoer:

```
I know your favorite number! It's 42.
```

## 10-13: Gebruiker verifiëren

```
import json

def get_stored_username():
    """Get stored username if available."""
    filename = 'username.json'
    try:
        with open(filename) as f_obj:
            username = json.load(f_obj)
    except FileNotFoundError:
        return None
    else:
        return username

def get_new_username():
    """Prompt for a new username."""
    username = input("What is your name? ")
    filename = 'username.json'
    with open(filename, 'w') as f_obj:
        json.dump(username, f_obj)
    return username

def greet_user():
    """Greet the user by name."""
    username = get_stored_username()
    if username:
        correct = input("Are you " + username + "? (y/n) ")
        if correct == 'y':
            print("Welcome back, " + username + "!")
        else:
            username = get_new_username()
            print("We'll remember you when you come back, " + username + "!")
    else:
        username = get_new_username()
        print("We'll remember you when you come back, " + username + "!")

greet_user()
```

Output:

```
> python verify_user.py
What is your name? eric
We'll remember you when you come back, eric!

> python verify_user.py
Are you eric? (y/n) y
Welcome back, eric!

> python verify_user.py
Are you eric? (y/n) n
What is your name? ever
We'll remember you when you come back, ever!

> python verify_user.py
Are you ever? (y/n) y
Welcome back, ever!
```

Mogelijk ziet u dezelfde identieke blokken in deze versie van `greet_user()`. Een manier om deze functie op te schonen, is door een leeg `return` statement te gebruiken. Een lege `return` statement vertelt Python om de functie te verlaten zonder nog meer code in de functie uit te voeren.

Hier is een schonere versie van `greet_user()`:

```
def greet_user():
    """Greet the user by name."""
    username = get_stored_username()
    if username:
        correct = input("Are you " + username + "? (y/n) ")
        if correct == 'y':
            print("Welcome back, " + username + "!")
            return

    # We got a username, but it's not correct.
    # Let's prompt for a new username.
    username = get_new_username()
    print("We'll remember you when you come back, " + username + "!")
```

De `return`-instructie betekent dat de code in de functie niet meer werkt nadat het welkomstbericht is afgedrukt. Wanneer de gebruikersnaam niet bestaat of de gebruikersnaam is onjuist, wordt de `return`-instructie nooit bereikt. Het tweede deel van de functie wordt alleen uitgevoerd als de `if`-instructies falen, dus we hebben geen ander blok nodig. Nu vraagt de functie om een nieuwe gebruikersnaam wanneer een van beide `if`-instructies falen.

Het enige dat overblijft om aan te pakken zijn de geneste if-instructies. Dit kan worden opgeschoond door de code te verplaatsen die controleert of de gebruikersnaam correct is voor een afzonderlijke functie. Als je deze oefening leuk vindt, kun je proberen een nieuwe functie met de naam `check_username()` aan te maken en te kijken of je de geneste `if`-instructie van `greet_user()` kunt verwijderen.

## 11-1: Stad, land

city_functions.py:

```
"""A collection of functions for working with cities."""

def city_country(city, country):
    """Return a string like 'Santiago, Chile'."""
    return(city.title() + ", " + country.title())
```

**Let op:** Dit is de functie die we schreven in Oefening 8-6.

test_cities.py:

```
import unittest

from city_functions import city_country

class CitiesTestCase(unittest.TestCase):
    """Tests for 'city_functions.py'."""

    def test_city_country(self):
        """Does a simple city and country pair work?"""
        santiago_chile = city_country('santiago', 'chile')
        self.assertEqual(santiago_chile, 'Santiago, Chile')

unittest.main()
```

Output:

```
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

## 11-2: Bevolking

Gewijzigde *city_functions.py*, met vereiste `population`-parameter:

```
"""A collection of functions for working with cities."""

def city_country(city, country, population):
    """Return a string like 'Santiago, Chile - population 5000000'."""
    output_string = city.title() + ", " + country.title()
    output_string += ' - population ' + str(population)
    return output_string
```

Output van het draaien van *test_cities.py:*

```
E
======================================================================
ERROR: test_city_country (__main__.CitiesTestCase)
Does a simple city and country pair work?
----------------------------------------------------------------------
Traceback (most recent call last):
  File "pcc\solutions\test_cities.py", line 10, in test_city_country
    santiago_chile = city_country('santiago', 'chile')
TypeError: city_country() missing 1 required positional argument: 'population'

----------------------------------------------------------------------
Ran 1 test in 0.000s

FAILED (errors=1)
```

Gewijzigde *city_functions.py*, met optionele `population`-parameter:

```
"""A collection of functions for working with cities."""

def city_country(city, country, population=0):
    """Return a string representing a city-country pair."""

    output_string = city.title() + ", " + country.title()
    if population:
        output_string += ' - population ' + str(population)
    return output_string
```

Output van het draaien van *test_cities.py:*

```
.
----------------------------------------------------------------------
Ran 1 test in 0.001s

OK
```

Gewijzigde *test_cities.py*:

```
import unittest

from city_functions import city_country

class CitiesTestCase(unittest.TestCase):
    """Tests for 'city_functions.py'."""

    def test_city_country(self):
```

```
        """Does a simple city and country pair work?"""
        santiago_chile = city_country('santiago', 'chile')
        self.assertEqual(santiago_chile, 'Santiago, Chile')

    def test_city_country_population(self):
        """Can I include a population argument?"""
        santiago_chile = city_country('santiago', 'chile', population=5000000)
        self.assertEqual(santiago_chile, 'Santiago, Chile - population 5000000')

unittest.main()
```

Output:

```
..
----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
```

## 11-3: Medewerker

employee.py:

```
class Employee():
    """A class to represent an employee."""

    def __init__(self, f_name, l_name, salary):
        """Initialize the employee."""
        self.first = f_name.title()
        self.last = l_name.title()
        self.salary = salary

    def give_raise(self, amount=5000):
        """Give the employee a raise."""
        self.salary += amount
```

test_employee.py:

```
import unittest

from employee import Employee

class TestEmployee(unittest.TestCase):
    """Tests for the module employee."""

    def setUp(self):
        """Make an employee to use in tests."""
        self.eric = Employee('eric', 'matthes', 65000)

    def test_give_default_raise(self):
        """Test that a default raise works correctly."""
        self.eric.give_raise()
        self.assertEqual(self.eric.salary, 70000)

    def test_give_custom_raise(self):
        """Test that a custom raise works correctly."""
        self.eric.give_raise(10000)
        self.assertEqual(self.eric.salary, 75000)

unittest.main()
```

Output:

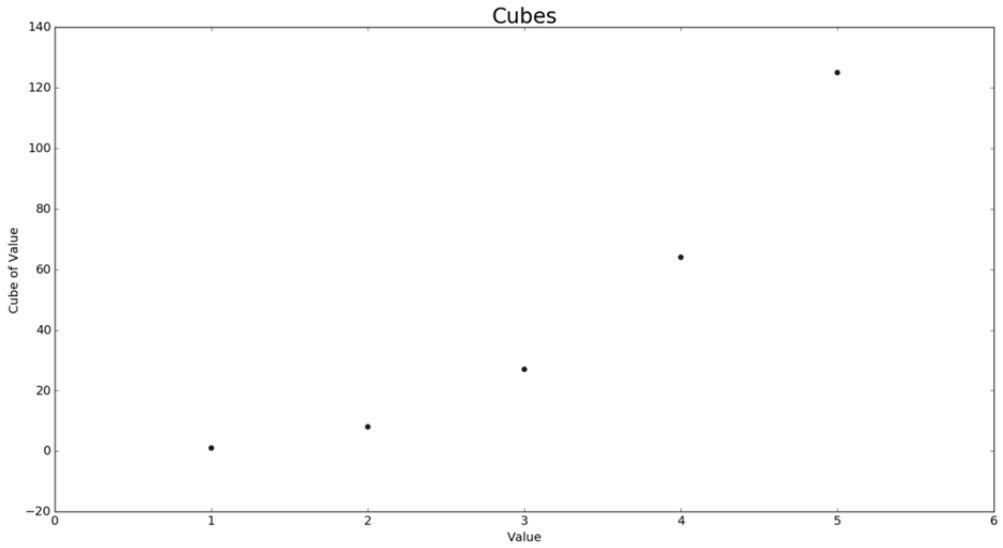
```
..
----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
```

## 15-1: Tot de derde macht

De eerste vijf getallen:

```
from matplotlib import pyplot as plt

# Define data.
x_values = [1, 2, 3, 4, 5]
cubes = [1, 8, 27, 64, 125]

# Make plot.
plt.scatter(x_values, cubes, edgecolor='none', s=40)

# Customize plot.
plt.title("Cubes", fontsize=24)
plt.xlabel('Value', fontsize=14)
plt.ylabel('Cube of Value', fontsize=14)
plt.tick_params(axis='both', labelsize=14)

# Show plot.
plt.show()
```

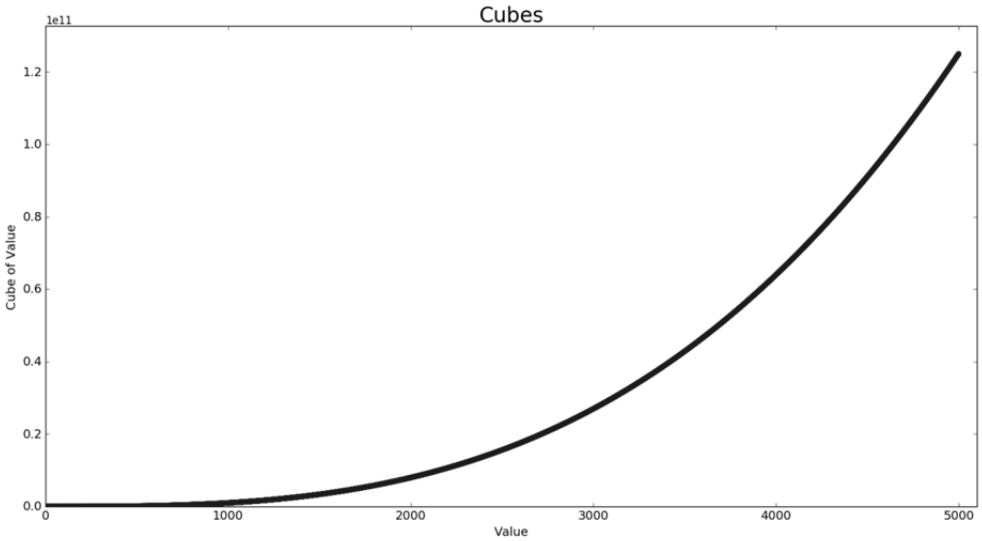
Output:




5000 getallen:

```
from matplotlib import pyplot as plt

# Define data.
x_values = list(range(5001))
cubes = [x**3 for x in x_values]

# Make plot.
plt.scatter(x_values, cubes, edgecolor='none', s=40)

# Customize plot.
plt.title("Cubes", fontsize=24)
plt.xlabel('Value', fontsize=14)
plt.ylabel('Cube of Value', fontsize=14)
plt.tick_params(axis='both', labelsize=14)
plt.axis([0, 5100, 0, 5100**3])

# Show plot.
plt.show()
```

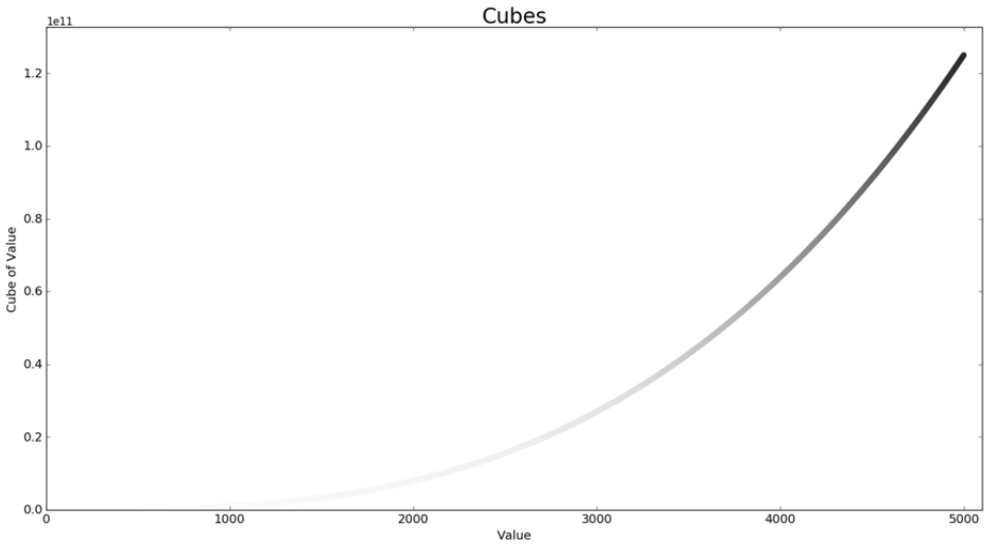
Output:




## 15-2: Gekleurde machten

```
from matplotlib import pyplot as plt

# Define data.
x_values = list(range(5001))
cubes = [x**3 for x in x_values]

# Make plot.
plt.scatter(x_values, cubes, edgecolor='none', c=cubes, cmap=plt.cm.BuGn, s=40)

# Customize plot.
plt.title("Cubes", fontsize=24)
plt.xlabel('Value', fontsize=14)
plt.ylabel('Cube of Value', fontsize=14)
plt.tick_params(axis='both', labelsize=14)
plt.axis([0, 5100, 0, 5100**3])

# Show plot.
plt.show()
```

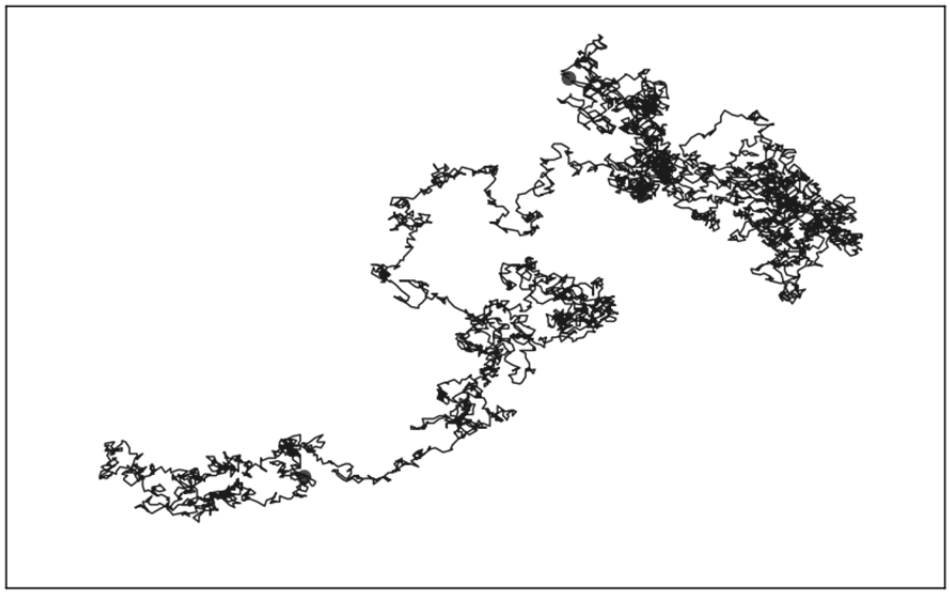
Output:



## 15-3: Moleculaire beweging

```
import matplotlib.pyplot as plt

from random_walk import RandomWalk

# Keep making new walks, as long as the program is active.
while True:
    # Make a random walk, and plot the points.
    rw = RandomWalk(5000)
    rw.fill_walk()

    # Set the size of the plotting window.
    plt.figure(dpi=128, figsize=(10, 6))

    point_numbers = list(range(rw.num_points))
    plt.plot(rw.x_values, rw.y_values, linewidth=1)

    # Emphasize the first and last points.
    plt.scatter(0, 0, c='green', edgecolors='none', s=75)
    plt.scatter(rw.x_values[-1], rw.y_values[-1], c='red', edgecolors='none',
        s=75)

    # Remove the axes.
    plt.axes().get_xaxis().set_visible(False)
    plt.axes().get_yaxis().set_visible(False)

    plt.show()

    keep_running = input("Make another walk? (y/n): ")
    if keep_running == 'n':
        break
```

Output:

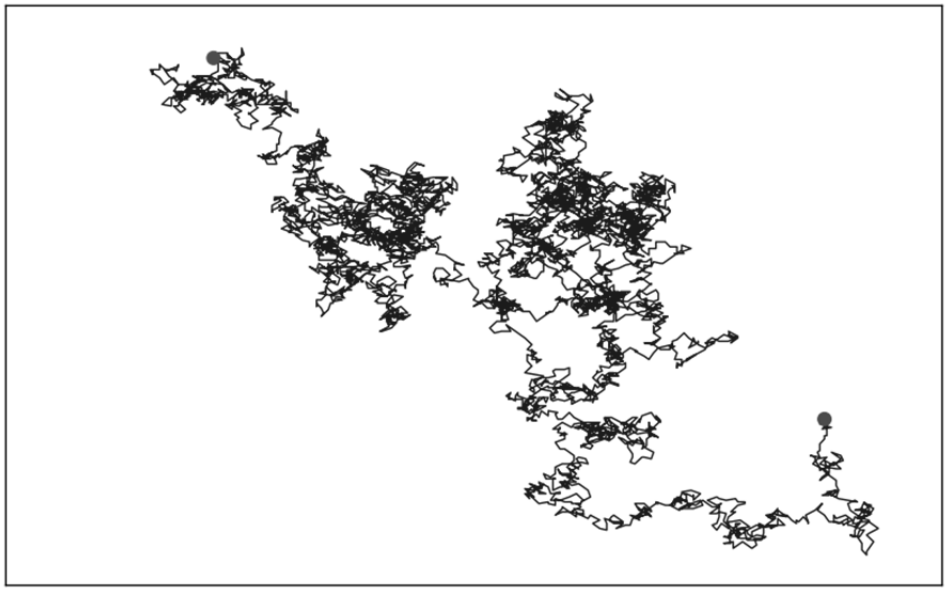


De spreidingsdiagrammen verschijnen achter de regels. Om ze boven de lijnen te plaatsen, kunnen we het `zorder`-argument gebruiken. Plotelementen met hogere `zorder`-waarden worden bovenop elementen met lagere `zorder`-waarden geplaatst.

```
import matplotlib.pyplot as plt

from random_walk import RandomWalk
```

```
# Keep making new walks, as long as the program is active.
while True:
    # Make a random walk, and plot the points.
    rw = RandomWalk(5000)
    rw.fill_walk()

    # Set the size of the plotting window.
    plt.figure(dpi=128, figsize=(10, 6))

    point_numbers = list(range(rw.num_points))
    plt.plot(rw.x_values, rw.y_values, linewidth=1, zorder=1)

    # Emphasize the first and last points.
    plt.scatter(0, 0, c='green', edgecolors='none', s=75, zorder=2)
    plt.scatter(rw.x_values[-1], rw.y_values[-1], c='red', edgecolors='none',
        s=75, zorder=2)

    # Remove the axes.
    plt.axes().get_xaxis().set_visible(False)
    plt.axes().get_yaxis().set_visible(False)

    plt.show()

    keep_running = input("Make another walk? (y/n): ")
    if keep_running == 'n':
        break
```

Output:



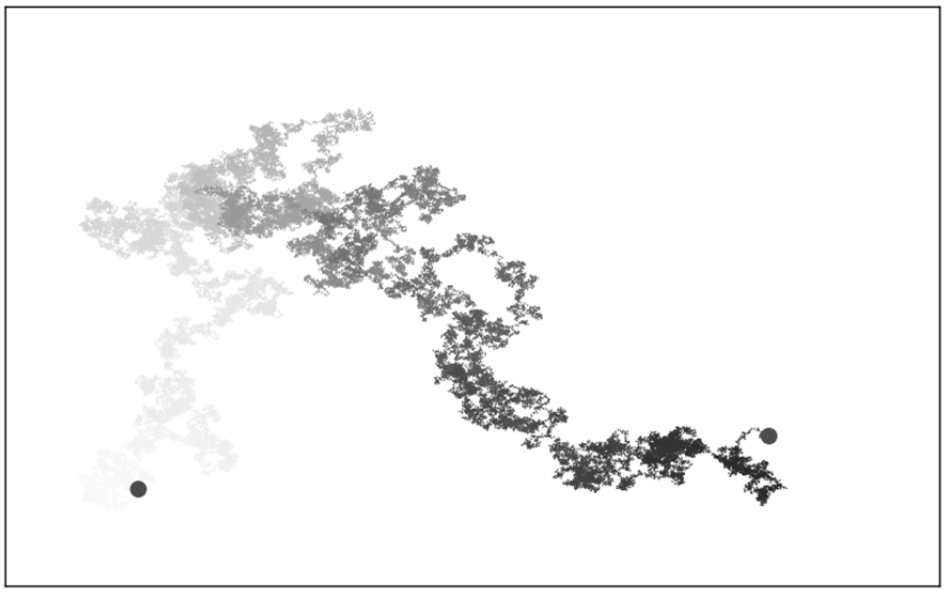
## 15-5: Opschonen

random_walk.py:

```
from random import choice

class RandomWalk():
    """A class to generate random walks."""

    def __init__(self, num_points=5000):
        """Initialize attributes of a walk."""
        self.num_points = num_points
```

```
        # All walks start at (0, 0).
        self.x_values = [0]
        self.y_values = [0]

    def get_step(self):
        """Determine the direction and distance for a step."""
        direction = choice([1, -1])
        distance = choice([0, 1, 2, 3, 4])
        step = direction * distance
        return step

    def fill_walk(self):
        """Calculate all the points in the walk."""

        # Keep taking steps until the walk reaches the desired length.
        while len(self.x_values) < self.num_points:

            # Decide which direction to go, and how far to go in that direction.
            x_step = self.get_step()
            y_step = self.get_step()

            # Reject moves that go nowhere.
            if x_step == 0 and y_step == 0:
                continue

            # Calculate the next x and y values.
            next_x = self.x_values[-1] + x_step
            next_y = self.y_values[-1] + y_step

            self.x_values.append(next_x)
            self.y_values.append(next_y)
```

Output:



## 15-6: Automatische labels

die_visual.py:

```
import pygal

from die import Die
```

```
# Create a D6.
die = Die()

# Make some rolls, and store results in a list.
results = [die.roll() for roll_num in range(1000)]

# Analyze the results.
frequencies = [results.count(value) for value in range(1, die.num_sides+1)]

# Visualize the results.
hist = pygal.Bar()

hist.title = "Results of rolling one D6 1000 times."
hist.x_labels = [str(x) for x in range(1, die.num_sides+1)]
hist.x_title = "Result"
hist.y_title = "Frequency of Result"

hist.add('D6', frequencies)
hist.render_to_file('die_visual.svg')
```

dice_visual.py:

```
import pygal

from die import Die

# Create two D6 dice.
die_1 = Die()
die_2 = Die()

# Make some rolls, and store results in a list.
results = [die_1.roll() + die_2.roll() for roll_num in range(1000)]

# Analyze the results.
max_result = die_1.num_sides + die_2.num_sides
frequencies = [results.count(value) for value in range(2, max_result+1)]

# Visualize the results.
hist = pygal.Bar()

hist.title = "Results of rolling two D6 dice 1000 times."
hist.x_labels = [str(x) for x in range(2, max_result+1)]
hist.x_title = "Result"
hist.y_title = "Frequency of Result"

hist.add('D6 + D6', frequencies)
hist.render_to_file('dice_visual.svg')
```

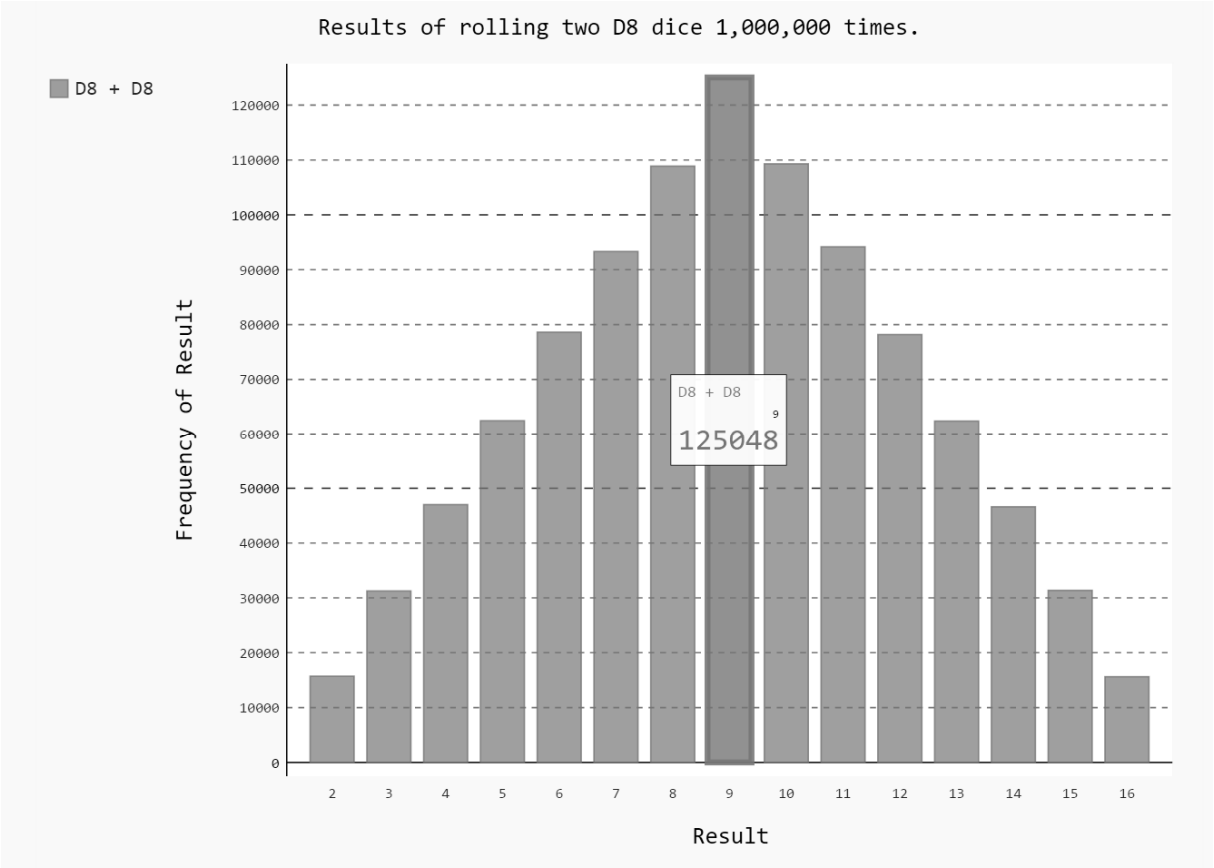
## 15-7: Twee keer een D8

```
import pygal

from die import Die

# Create two D8 dice.
die_1 = Die(8)
die_2 = Die(8)

# Make some rolls, and store results in a list.
results = []
for roll_num in range(1000000):
    result = die_1.roll() + die_2.roll()
    results.append(result)

# Analyze the results.
frequencies = []
max_result = die_1.num_sides + die_2.num_sides
for value in range(2, max_result+1):
    frequency = results.count(value)
```

```
    frequencies.append(frequency)

# Visualize the results.
hist = pygal.Bar()

hist.title = "Results of rolling two D8 dice 1,000,000 times."
hist.x_labels = [str(x) for x in range(2, max_result+1)]
hist.x_title = "Result"
hist.y_title = "Frequency of Result"

hist.add('D8 + D8', frequencies)
hist.render_to_file('dice_visual.svg')
```

Let op: deze oplossing gebruikt alleen een lijstcomprehensie voor de parameter `hist.x_labels`. Je zou ook kunnen proberen de andere lussen te vervangen door comprehensies.

Output:



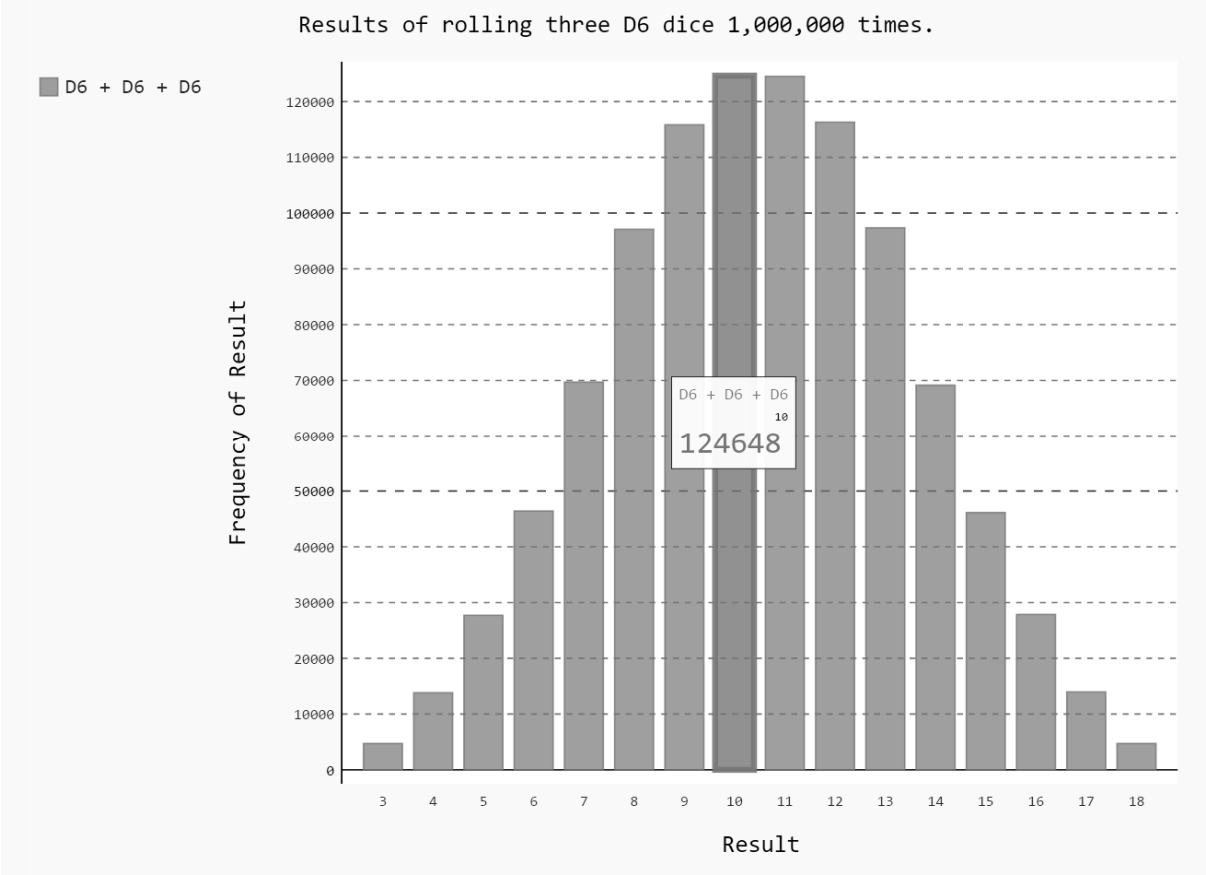

## 15-8: Drie dobbelstenen

```
import pygal

from die import Die

# Create three D6 dice.
die_1 = Die()
die_2 = Die()
die_3 = Die()

# Make some rolls, and store results in a list.
results = []
for roll_num in range(1000000):
    result = die_1.roll() + die_2.roll() + die_3.roll()
    results.append(result)

# Analyze the results.
```

```
frequencies = []
max_result = die_1.num_sides + die_2.num_sides + die_3.num_sides
for value in range(3, max_result+1):
    frequency = results.count(value)
    frequencies.append(frequency)

# Visualize the results.
hist = pygal.Bar()

hist.title = "Results of rolling three D6 dice 1,000,000 times."
hist.x_labels = [str(x) for x in range(3, max_result+1)]
hist.x_title = "Result"
hist.y_title = "Frequency of Result"

hist.add('D6 + D6 + D6', frequencies)
hist.render_to_file('dice_visual.svg')
```

Let op: deze oplossing gebruikt alleen een lijstcomprehensie voor de parameter `hist.x_labels` . Je zou ook kunnen proberen de andere lussen te vervangen door comprehensies.

Output:



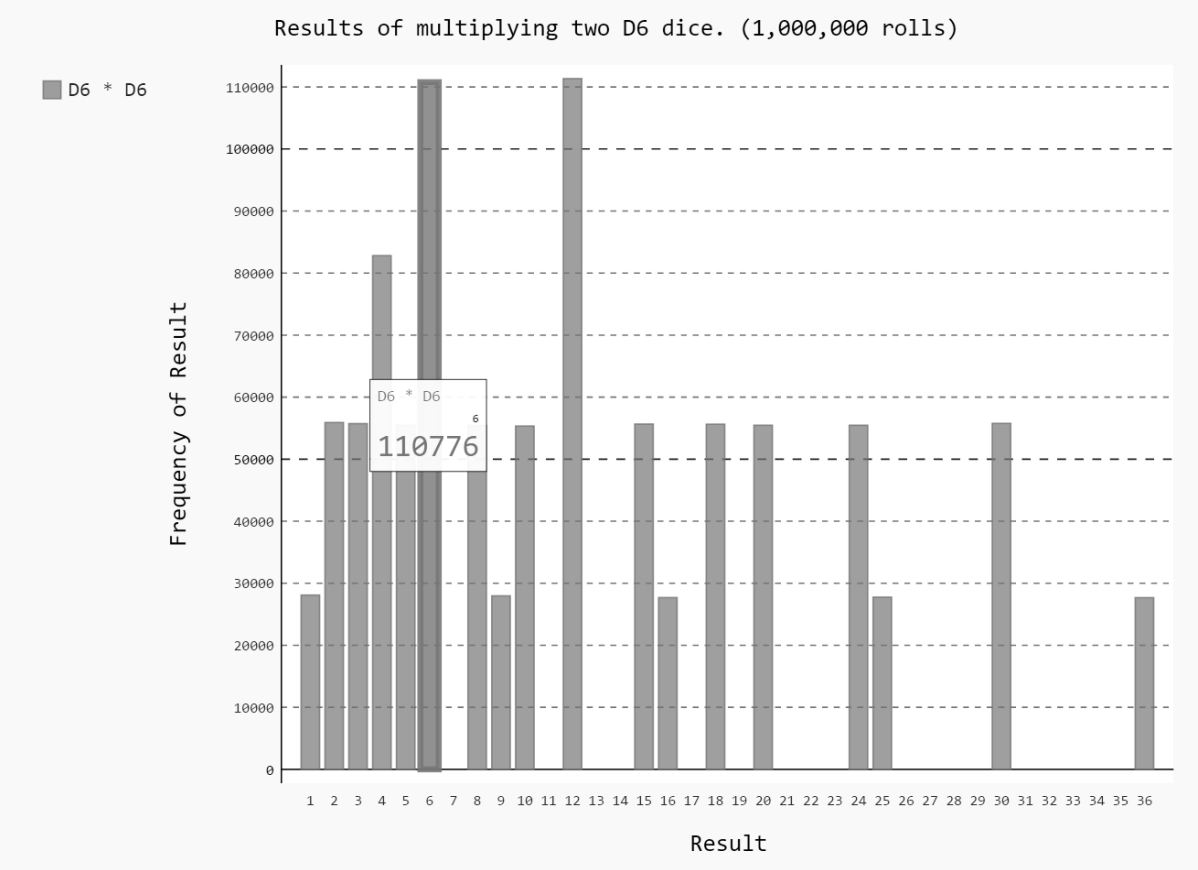

## 15-9: Vermenigvuldiging

```
import pygal

from die import Die

# Create two D6 dice.
die_1 = Die()
die_2 = Die()

# Make some rolls, and store results in a list.
results = []
for roll_num in range(1000000):
    result = die_1.roll() * die_2.roll()
```

```
    results.append(result)

# Analyze the results.
frequencies = []
max_result = die_1.num_sides * die_2.num_sides
for value in range(1, max_result+1):
    frequency = results.count(value)
    frequencies.append(frequency)

# Visualize the results.
hist = pygal.Bar()

hist.title = "Results of multiplying two D6 dice. (1,000,000 rolls)"
hist.x_labels = [str(x) for x in range(1, max_result+1)]
hist.x_title = "Result"
hist.y_title = "Frequency of Result"

hist.add('D6 * D6', frequencies)
hist.render_to_file('dice_visual.svg')
```

Let op: deze oplossing gebruikt alleen een lijstcomprehensie voor de parameter `hist.x_labels` . Je zou ook kunnen proberen de andere lussen te vervangen door comprehensies.

Output:



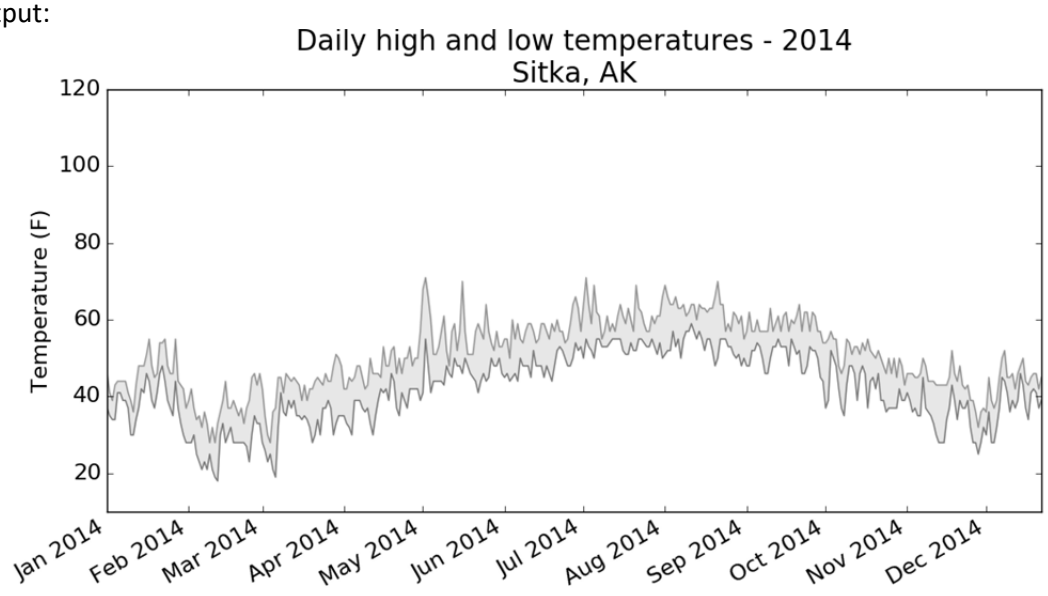

## 16-2: Sitka-Death Valley-vergelijking

Met de `pyplot` -functie `ylim()` kunt u de grenzen van alleen de y-as instellen. Als u ooit de limieten van de x-as moet opgeven, is er ook een bijbehorende `xlim()`-functie.

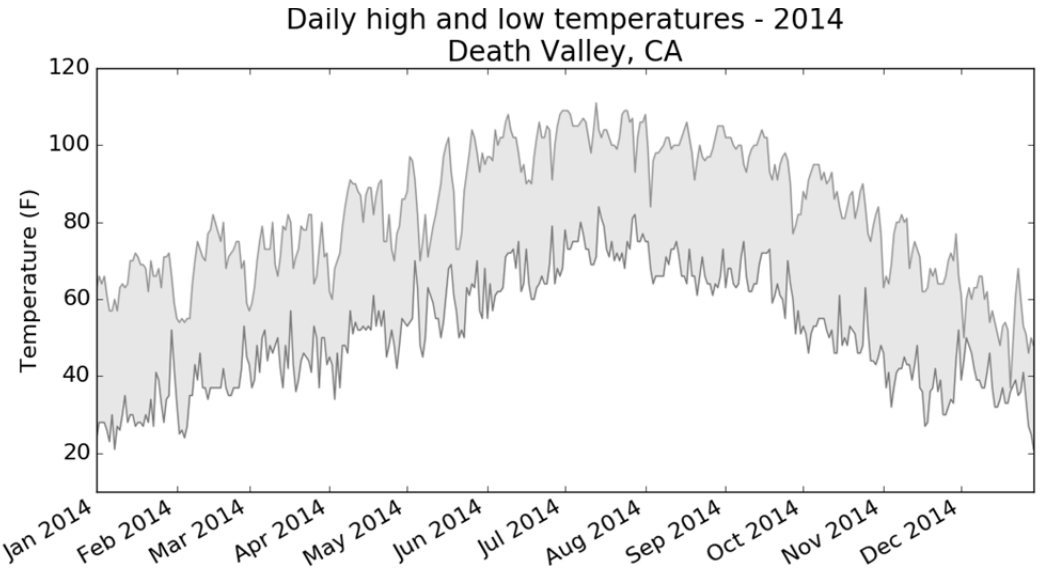
```
import csv
from datetime import datetime

from matplotlib import pyplot as plt
```

```
# Get dates, high, and low temperatures from file.
filename = 'sitka_weather_2014.csv'
with open(filename) as f:
    reader = csv.reader(f)
    header_row = next(reader)

    dates, highs, lows = [], [], []
    for row in reader:
        try:
            current_date = datetime.strptime(row[0], "%Y-%m-%d")
            high = int(row[1])
            low = int(row[3])
        except ValueError:
            print(current_date, 'missing data')
        else:
            dates.append(current_date)
            highs.append(high)
            lows.append(low)

# Plot data.
fig = plt.figure(dpi=128, figsize=(10, 6))
plt.plot(dates, highs, c='red', alpha=0.5)
plt.plot(dates, lows, c='blue', alpha=0.5)
plt.fill_between(dates, highs, lows, facecolor='blue', alpha=0.1)

# Format plot.
title = "Daily high and low temperatures - 2014\nSitka, AK"
plt.title(title, fontsize=20)
plt.xlabel('', fontsize=16)
fig.autofmt_xdate()
plt.ylabel("Temperature (F)", fontsize=16)
plt.tick_params(axis='both', which='major', labelsize=16)
plt.ylim(10, 120)

plt.show()
```

Output:



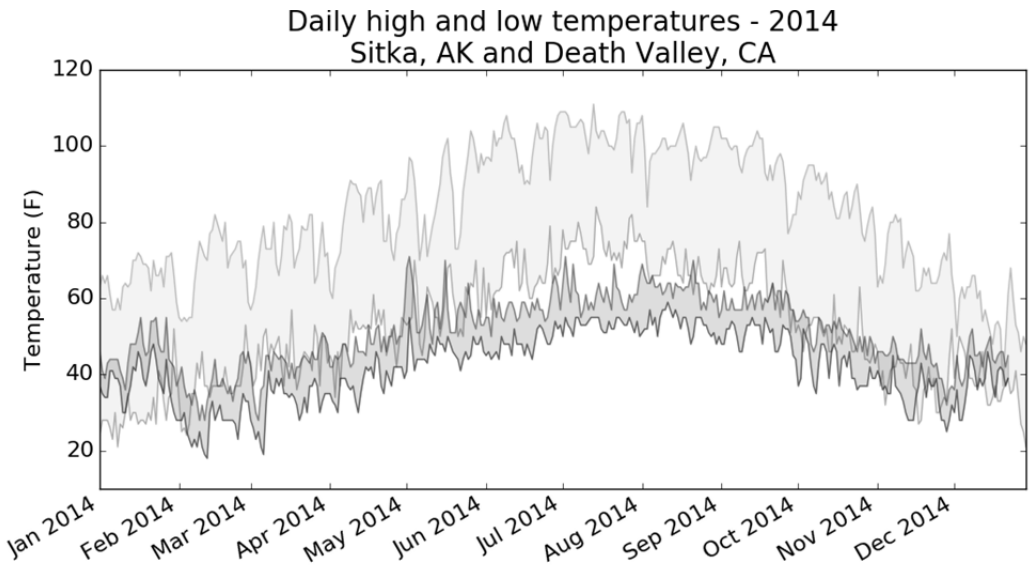
Het gebruik van dezelfde limieten voor de functie `ylim()` met de gegevens van Death Valley resulteert in een diagram met dezelfde schaal:




Er is een aantal manieren waarop je beide datasets in dezelfde grafiek kunt plotten. In de volgende oplossing plaatsen we de code voor het lezen van het csv-bestand in een functie. We noemen het dan een keer om de hoogtepunten en dieptepunten voor Sitka te krijgen voordat we de grafiek maken. Vervolgens roepen we de functie een tweede keer aan om de gegevens van Death Valley aan de bestaande plot toe te voegen. De kleuren zijn enigszins aangepast om de gegevens van elke locatie anders te maken.

```
import csv
from datetime import datetime

from matplotlib import pyplot as plt

def get_weather_data(filename, dates, highs, lows):
    """Get the highs and lows from a data file."""
    with open(filename) as f:
        reader = csv.reader(f)
        header_row = next(reader)

        # dates, highs, lows = [], [], []
        for row in reader:
            try:
                current_date = datetime.strptime(row[0], "%Y-%m-%d")
                high = int(row[1])
                low = int(row[3])
            except ValueError:
                print(current_date, 'missing data')
            else:
                dates.append(current_date)
                highs.append(high)
                lows.append(low)

# Get weather data for Sitka.
dates, highs, lows = [], [], []
get_weather_data('sitka_weather_2014.csv', dates, highs, lows)

# Plot Sitka weather data.
fig = plt.figure(dpi=128, figsize=(10, 6))
plt.plot(dates, highs, c='red', alpha=0.6)
plt.plot(dates, lows, c='blue', alpha=0.6)
plt.fill_between(dates, highs, lows, facecolor='blue', alpha=0.15)

# Get Death Valley data.
dates, highs, lows = [], [], []
```

```
get_weather_data('death_valley_2014.csv', dates, highs, lows)

# Add Death Valley data to current plot.
plt.plot(dates, highs, c='red', alpha=0.3)
plt.plot(dates, lows, c='blue', alpha=0.3)
plt.fill_between(dates, highs, lows, facecolor='blue', alpha=0.05)

# Format plot.
title = "Daily high and low temperatures - 2014"
title += "\nSitka, AK and Death Valley, CA"
plt.title(title, fontsize=20)
plt.xlabel('', fontsize=16)
fig.autofmt_xdate()
plt.ylabel("Temperature (F)", fontsize=16)
plt.tick_params(axis='both', which='major', labelsize=16)
plt.ylim(10, 120)

plt.show()
```



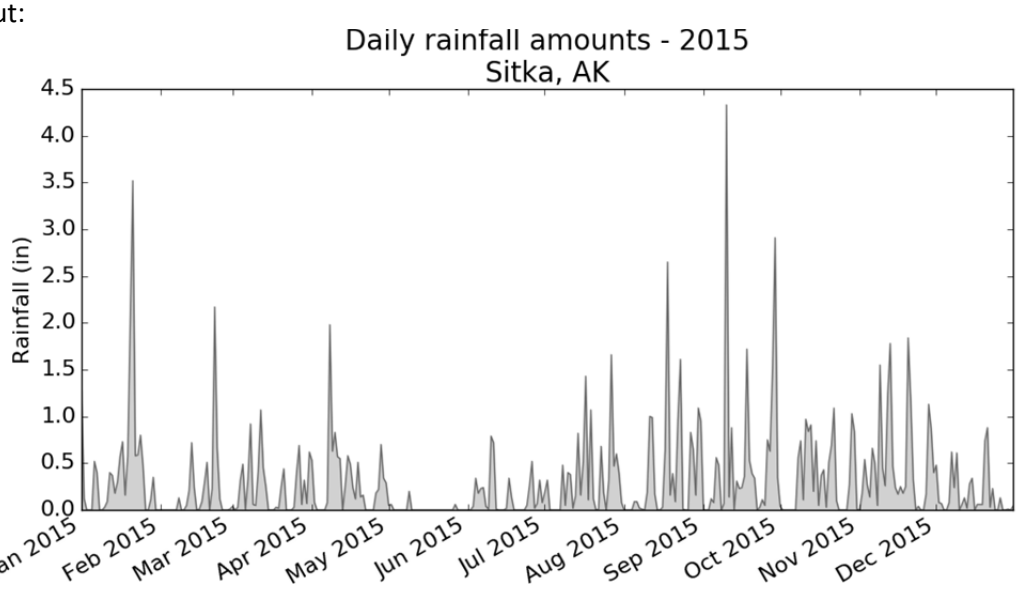

## 16-3: Neerslag

**Let op:** In dit voorbeeld is de data van deze pagina gebruikt.

```
import csv
from datetime import datetime

from matplotlib import pyplot as plt

# Get dates and rainfall data from data file.
#  Rainfall data is in column 19.
filename = 'sitka_rainfall_2015.csv'
with open(filename) as f:
    reader = csv.reader(f)
    header_row = next(reader)

    dates, rainfalls = [], []
    for row in reader:
        try:
            current_date = datetime.strptime(row[0], "%Y-%m-%d")
            rainfall = float(row[19])
        except ValueError:
            print(current_date, 'missing data')
        else:
            dates.append(current_date)
            rainfalls.append(rainfall)
```

```
# Plot data.
fig = plt.figure(dpi=128, figsize=(10, 6))
plt.plot(dates, rainfalls, c='blue', alpha=0.5)
plt.fill_between(dates, rainfalls, facecolor='blue', alpha=0.2)

# Format plot.
title = "Daily rainfall amounts - 2015\nSitka, AK"
plt.title(title, fontsize=20)
plt.xlabel('', fontsize=16)
fig.autofmt_xdate()
plt.ylabel("Rainfall (in)", fontsize=16)
plt.tick_params(axis='both', which='major', labelsize=16)

plt.show()
```

Output:



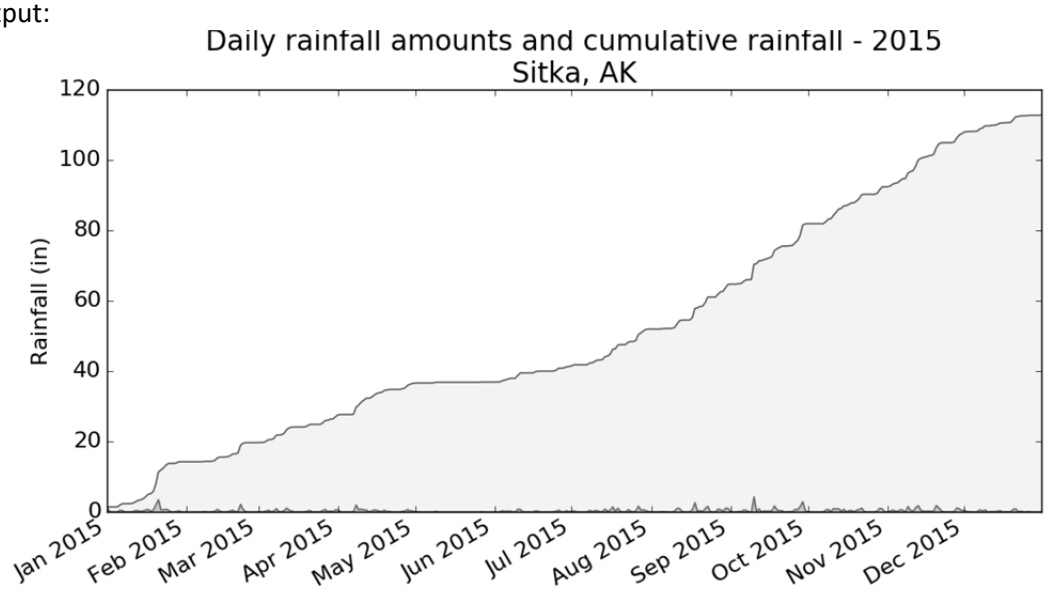

## 16-4: Verkennen

Ik woon in een regenwoud, dus ik was geïnteresseerd in de neerslaggegevens. Ik heb de cumulatieve regenval voor het jaar berekend en dat uitgezet tegen de dagelijkse regenval. Zelfs na het leven in deze regen, ben ik verrast om te zien hoeveel we krijgen.

```
import csv
from datetime import datetime

from matplotlib import pyplot as plt

# Get dates and rainfall data from data file.
#  Rainfall data is in column 19.
filename = 'sitka_rainfall_2015.csv'
with open(filename) as f:
    reader = csv.reader(f)
    header_row = next(reader)

    dates, rainfalls, totals = [], [], []
    for row in reader:
        try:
            current_date = datetime.strptime(row[0], "%Y-%m-%d")
            rainfall = float(row[19])
        except ValueError:
            print(current_date, 'missing data')
        else:
            dates.append(current_date)
            rainfalls.append(rainfall)
            if totals:
```

```
            totals.append(totals[-1] + rainfall)
        else:
            totals.append(rainfall)

# Plot data.
fig = plt.figure(dpi=128, figsize=(10, 6))
plt.plot(dates, rainfalls, c='blue', alpha=0.5)
plt.fill_between(dates, rainfalls, facecolor='blue', alpha=0.2)

plt.plot(dates, totals, c='blue', alpha=0.75)
plt.fill_between(dates, totals, facecolor='blue', alpha=0.05)

# Format plot.
title = "Daily rainfall amounts and cumulative rainfall - 2015\nSitka, AK"
plt.title(title, fontsize=20)
plt.xlabel('', fontsize=16)
fig.autofmt_xdate()
plt.ylabel("Rainfall (in)", fontsize=16)
plt.tick_params(axis='both', which='major', labelsize=16)

plt.show()
```

Output:




## 16-6: Bruto binnenlands product

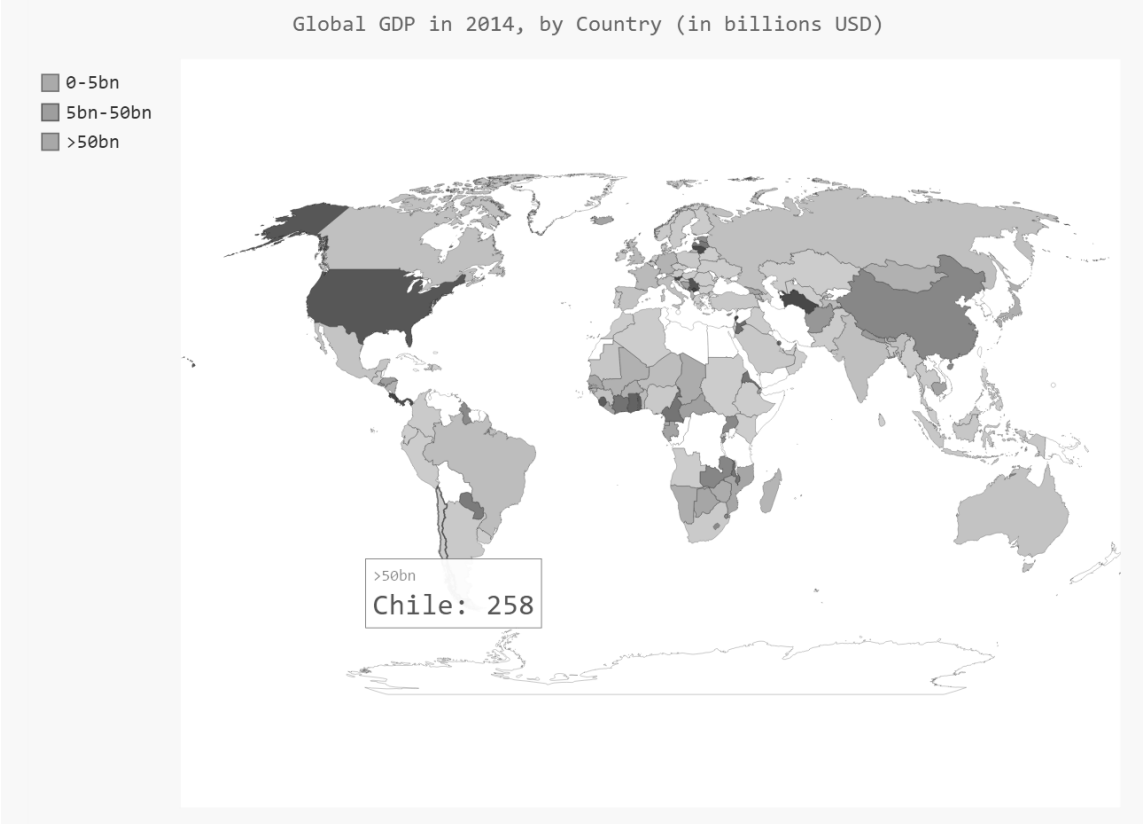
**Let op:** Als je problemen hebt met het downloaden van het JSON-bestand voor de data van het BBP, dan kan je deze link gebruiken. Als dat ook niet werkt, dan heb ik ook een kopie van het bestand hier opgeslagen.

***Let op:** Houd er rekening mee dat sommige versies van dit gegevensbestand al jaren tussen aanhalingstekens staan en sommige hebben de jaren niet vermeld. Wanneer de jaren worden geciteerd, worden ze als strings behandeld. Wanneer ze niet worden vermeld, worden ze behandeld als numerieke gegevens. Mogelijk moet je vergelijkingen wijzigen, zoals `if gdp_dict['Year'] == '2014'` to `if gdp_dict['Year'] == 2014:`.

```
import json

import pygal
from pygal.style import LightColorizedStyle as LCS, RotateStyle as RS
from pygal.maps.world import World

from country_codes import get_country_code
```

```
# Load the data into a list.
filename = 'global_gdp.json'
with open(filename) as f:
    gdp_data = json.load(f)

# Build a dictionary of gdp data.
cc_gdps = {}
for gdp_dict in gdp_data:
    if gdp_dict['Year'] == '2014':
        country_name = gdp_dict['Country Name']
        gdp = int(float(gdp_dict['Value']))
        code = get_country_code(country_name)
        if code:
            cc_gdps[code] = gdp

# Group the countries into 3 gdp levels.
#  Less than 5 billion, less than 50 billion, >= 50 billion.
#  Also, convert to billions for displaying values.
cc_gdps_1, cc_gdps_2, cc_gdps_3 = {}, {}, {}
for cc, gdp in cc_gdps.items():
    if gdp < 5000000000:
        cc_gdps_1[cc] = round(gdp / 1000000000)
    elif gdp < 50000000000:
        cc_gdps_2[cc] = round(gdp / 1000000000)
    else:
        cc_gdps_3[cc] = round(gdp / 1000000000)

# See how many countries are in each level.
print(len(cc_gdps_1), len(cc_gdps_2), len(cc_gdps_3))

wm_style = RS('#336699', base_style=LCS)
wm = World(style=wm_style)
wm.title = 'Global GDP in 2014, by Country (in billions USD)'
wm.add('0-5bn', cc_gdps_1)
wm.add('5bn-50bn', cc_gdps_2)
wm.add('>50bn', cc_gdps_3)

wm.render_to_file('global_gdp.svg')
```

Output:



## 16-8: Testen van de country_codes-module

```
import unittest

from country_codes import get_country_code

class CountryCodesTestCase(unittest.TestCase):
    """Tests for country_codes.py."""

    def test_get_country_code(self):
        country_code = get_country_code('Andorra')
        self.assertEqual(country_code, 'ad')

        country_code = get_country_code('United Arab Emirates')
        self.assertEqual(country_code, 'ae')

        country_code = get_country_code('Afghanistan')
        self.assertEqual(country_code, 'af')

unittest.main()
```

Output:

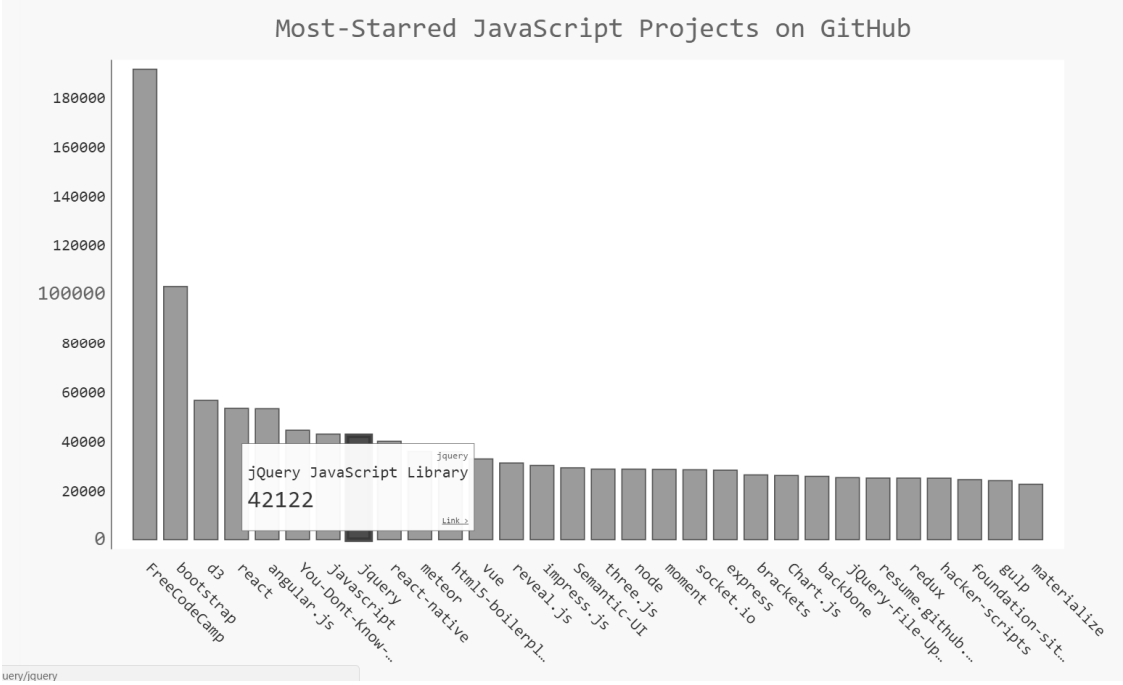
```
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

## 17-1: Andere talen

```
import requests
import pygal
from pygal.style import LightColorizedStyle as LCS, LightenStyle as LS

# Make an API call, and store the response.
url = 'https://api.github.com/search/repositories?q=language:javascript&sort=stars'
r = requests.get(url)
print("Status code:", r.status_code)

# Store API response in a variable.
response_dict = r.json()
print("Total repositories:", response_dict['total_count'])

# Explore information about the repositories.
repo_dicts = response_dict['items']

names, plot_dicts = [], []
for repo_dict in repo_dicts:
    names.append(repo_dict['name'])

    # When a project is removed, it's still listed with stars.
    #   So it's in the top projects, but has no description. The description
    #   is None, which causes an exception when being used as a label.
    if repo_dict['description']:
        desc = repo_dict['description']
    else:
        desc = 'No description provided.'

    plot_dict = {
        'value': repo_dict['stargazers_count'],
        'label': desc,
        'xlink': repo_dict['html_url'],
        }
    plot_dicts.append(plot_dict)

# Make visualization.
my_style = LS('#333366', base_style=LCS)
```

```
my_style.title_font_size = 24
my_style.label_font_size = 14
my_style.major_label_font_size = 18

my_config = pygal.Config()
my_config.x_label_rotation = 45
my_config.show_legend = False
my_config.truncate_label = 15
my_config.show_y_guides = False
my_config.width = 1000

chart = pygal.Bar(my_config, style=my_style)
chart.title = 'Most-Starred JavaScript Projects on GitHub'
chart.x_labels = names

chart.add('', plot_dicts)
chart.render_to_file('js_repos.svg')
```

Output:

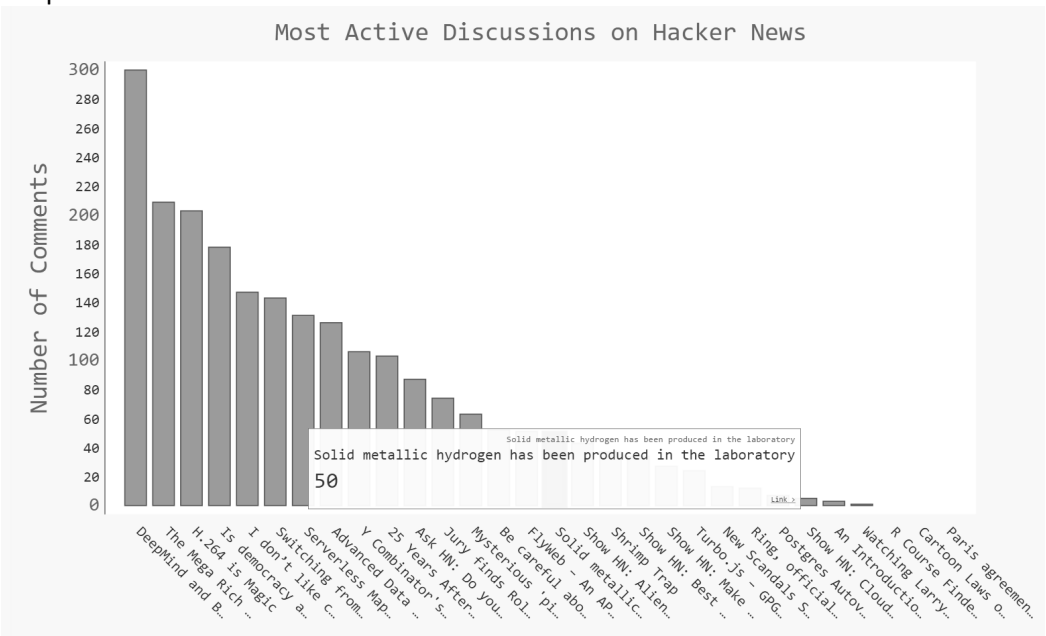



## 17-2: Andere discussies

```
import requests
import pygal
from pygal.style import LightColorizedStyle as LCS, LightenStyle as LS

from operator import itemgetter

# Make an API call, and store the response.
url = 'https://hacker-news.firebaseio.com/v0/topstories.json'
r = requests.get(url)
print("Status code:", r.status_code)

# Process information about each submission.
submission_ids = r.json()
submission_dicts = []
for submission_id in submission_ids[:30]:
    # Make a separate API call for each submission.
    url = ('https://hacker-news.firebaseio.com/v0/item/' +
            str(submission_id) + '.json')
    submission_r = requests.get(url)
    print(submission_r.status_code)
    response_dict = submission_r.json()
```

```
    submission_dict = {
        'title': response_dict['title'],
        'link': 'http://news.ycombinator.com/item?id=' + str(submission_id),
        'comments': response_dict.get('descendants', 0)
        }
    submission_dicts.append(submission_dict)

submission_dicts = sorted(submission_dicts, key=itemgetter('comments'),
                          reverse=True)

for submission_dict in submission_dicts:
    print("\nTitle:", submission_dict['title'])
    print("Discussion link:", submission_dict['link'])
    print("Comments:", submission_dict['comments'])

titles, plot_dicts = [], []
for submission_dict in submission_dicts:
    titles.append(submission_dict['title'])
    plot_dict = {
        'value': submission_dict['comments'],
        'label': submission_dict['title'],
        'xlink': submission_dict['link'],
        }
    plot_dicts.append(plot_dict)

# Make visualization.
my_style = LS('#333366', base_style=LCS)
my_style.title_font_size = 24
my_style.label_font_size = 14
my_style.major_label_font_size = 18

my_config = pygal.Config()
my_config.x_label_rotation = 45
my_config.show_legend = False
my_config.truncate_label = 15
my_config.show_y_guides = False
my_config.width = 1000
my_config.y_title = 'Number of Comments'

chart = pygal.Bar(my_config, style=my_style)
chart.title = 'Most Active Discussions on Hacker News'
chart.x_labels = titles

chart.add('', plot_dicts)
chart.render_to_file('hn_discussions.svg')
```

Output:



## 17-3: Testen van python_repos.py

```
import requests
import pygal
from pygal.style import LightColorizedStyle as LCS, LightenStyle as LS

def get_response():
    """Make an api call, and return the response."""
    url = 'https://api.github.com/search/repositories?q=language:python&sort=stars'
    r = requests.get(url)
    return r

def get_repo_dicts(response):
    """Return a set of dicts representing the most popular repositories."""
    response_dict = r.json()
    repo_dicts = response_dict['items']
    return repo_dicts

def get_names_plot_dicts(repo_dicts):
    """Process the set of repository dicts, and pull out data for plotting."""
    names, plot_dicts = [], []
    for repo_dict in repo_dicts:
        names.append(repo_dict['name'])

        # Some projects lack a description, which causes an error when
        #  labeling bars. Specify a label if there's no description.
        description = repo_dict['description']
        if not description:
            description = "No description provided."

        plot_dict = {
            'value': repo_dict['stargazers_count'],
            'label': description,
            'xlink': repo_dict['html_url'],
            }
        plot_dicts.append(plot_dict)
    return names, plot_dicts

def make_visualization(names, plot_dicts):
    """Make visualization of most popular repositories."""
    my_style = LS('#333366', base_style=LCS)
    my_style.title_font_size = 24
    my_style.label_font_size = 14
    my_style.major_label_font_size = 18

    my_config = pygal.Config()
    my_config.x_label_rotation = 45
    my_config.show_legend = False
    my_config.truncate_label = 15
    my_config.show_y_guides = False
    my_config.width = 1000

    chart = pygal.Bar(my_config, style=my_style)
    chart.title = 'Most-Starred Python Projects on GitHub'
    chart.x_labels = names

    chart.add('', plot_dicts)
    chart.render_to_file('python_repos.svg')


r = get_response()
repo_dicts = get_repo_dicts(r)
names, plot_dicts = get_names_plot_dicts(repo_dicts)
make_visualization(names, plot_dicts)
```

Nu kunnen we tests voor deze functies schrijven. Hier testen we dat we een antwoord krijgen met een statuscode van 200 en we testen dat enkele van de sleutels die we verwachten te vinden in het woordenboek van elke repository zich in het woordenboek van het eerste project bevinden.

```
import unittest

import python_repos_for_testing as pr

class PythonReposTestCase(unittest.TestCase):
    """Tests for python_repos.py."""

    def setUp(self):
        """Call all the functions here, and test elements separately."""
        self.r = pr.get_response()
        self.repo_dicts = pr.get_repo_dicts(self.r)
        self.repo_dict = self.repo_dicts[0]
        self.names, self.plot_dicts = pr.get_names_plot_dicts(self.repo_dicts)

    def test_get_response(self):
        """Test that we get a valid response."""
        self.assertEqual(self.r.status_code, 200)

    def test_repo_dicts(self):
        """Test that we're getting the data we think we are."""
        # We should get dicts for 30 repositories.
        self.assertEqual(len(self.repo_dicts), 30)

        # Repositories should have required keys.
        required_keys = ['name', 'owner', 'stargazers_count', 'html_url']
        for key in required_keys:
            self.assertTrue(key in self.repo_dict.keys())

unittest.main()
```

Output:

```
..
----------------------------------------------------------------------
Ran 2 tests in 1.969s

OK
```