GEACTUALISEERD
11 NOV 2026



7 ITERIOLEN IN PYTHON

Meindert Meindertsma — 22 januari & 28 september 2024

Spelers en ‘happy flows’ in het iteratiespel

Wat doet het **for**-statement?

```
for <pakketje> in <bron>:
    <handelingen>
```

- Eenmalig: methode *<bron>*.**__iter__** aanroepen om een iterator *<iterator>* op te halen.
- Herhaald: methode *<iterator>*.**__next__** aanroepen om een nieuw *<pakketje>* op te halen en vervolgens *<handelingen>* verrichten.
- Stoppen wanneer *<iterator>*.**__next__()** een **StopIteration**-exception genereert.

Wat doet de **iter**-functie?

```
iter(<bron>)
```

- Methode *<bron>*.**__iter__** aanroepen om een iterator *<iterator>* op te halen.
- Controleren of dit echt een iterator is, d.w.z. een methode **__next__** heeft.
- Als het verkregen object echt een iterator is, deze als returnwaarde doorgeven aan de aanroeper.

Wat doet de **next**-functie?

```
next(<iterator>[,<default>])
```

- Methode *<iterator>*.**__next__** aanroepen om een nieuw *<pakketje>* op te halen.
- Het verkregen *<pakketje>* als returnwaarde doorgeven aan de aanroeper.
- Wanneer *<iterator>*.**__next__()** een **StopIteration**-exception genereert, deze doorspelen naar de aanroeper, tenzij *<default>* gespecificeerd is (in dat geval *<default>* als returnwaarde doorgeven aan de aanroeper).

Wat behoort een **__iter__**-methode te doen?

```
<bron>.__iter__()
```

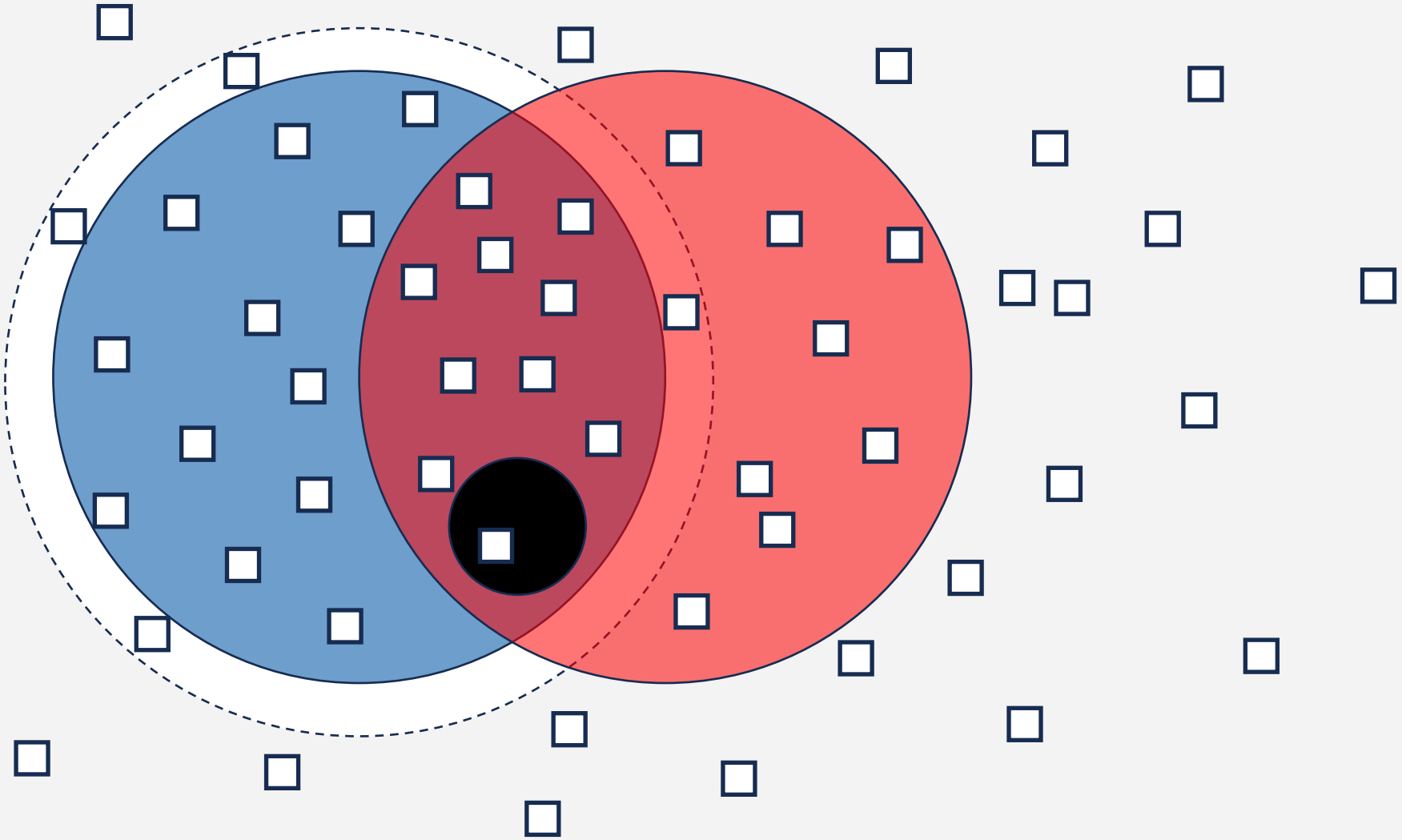
- Een iterator *<iterator>* als returnwaarde doorgeven aan de aanroeper.

Wat behoort een **__next__**-methode te doen?

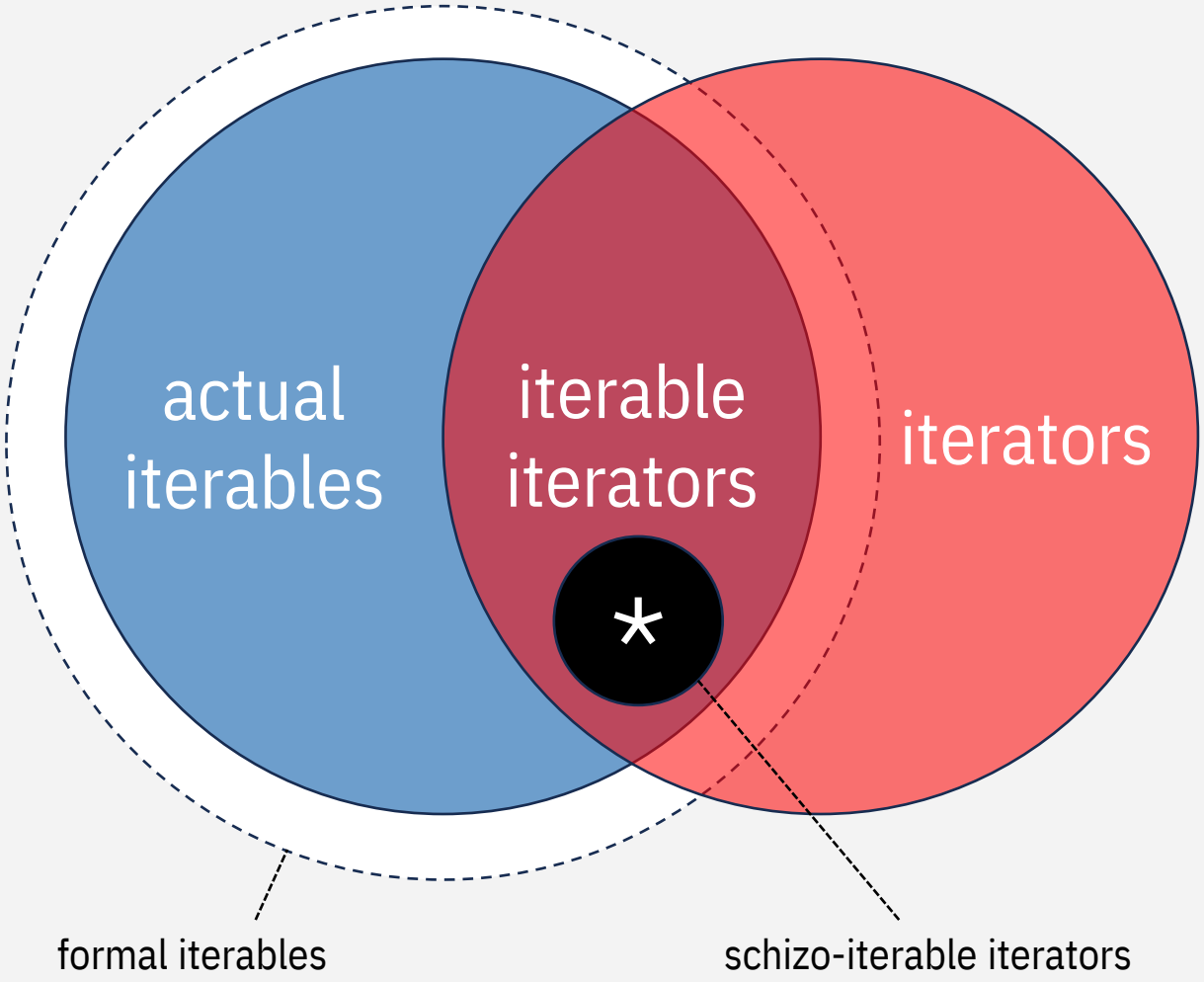
```
<iterator>.__next__()
```

- Als dat te leveren is, een nieuw *<pakketje>* als returnwaarde doorgeven aan de aanroeper.
- Anders een **StopIteration**-exception genereren.

Eulerdiagrammen (ieder blokje is een object)



Eulerdiagram van **‘iterables’** en **‘iterators’**



Te gebruiken achter **in** in **for**-statement:



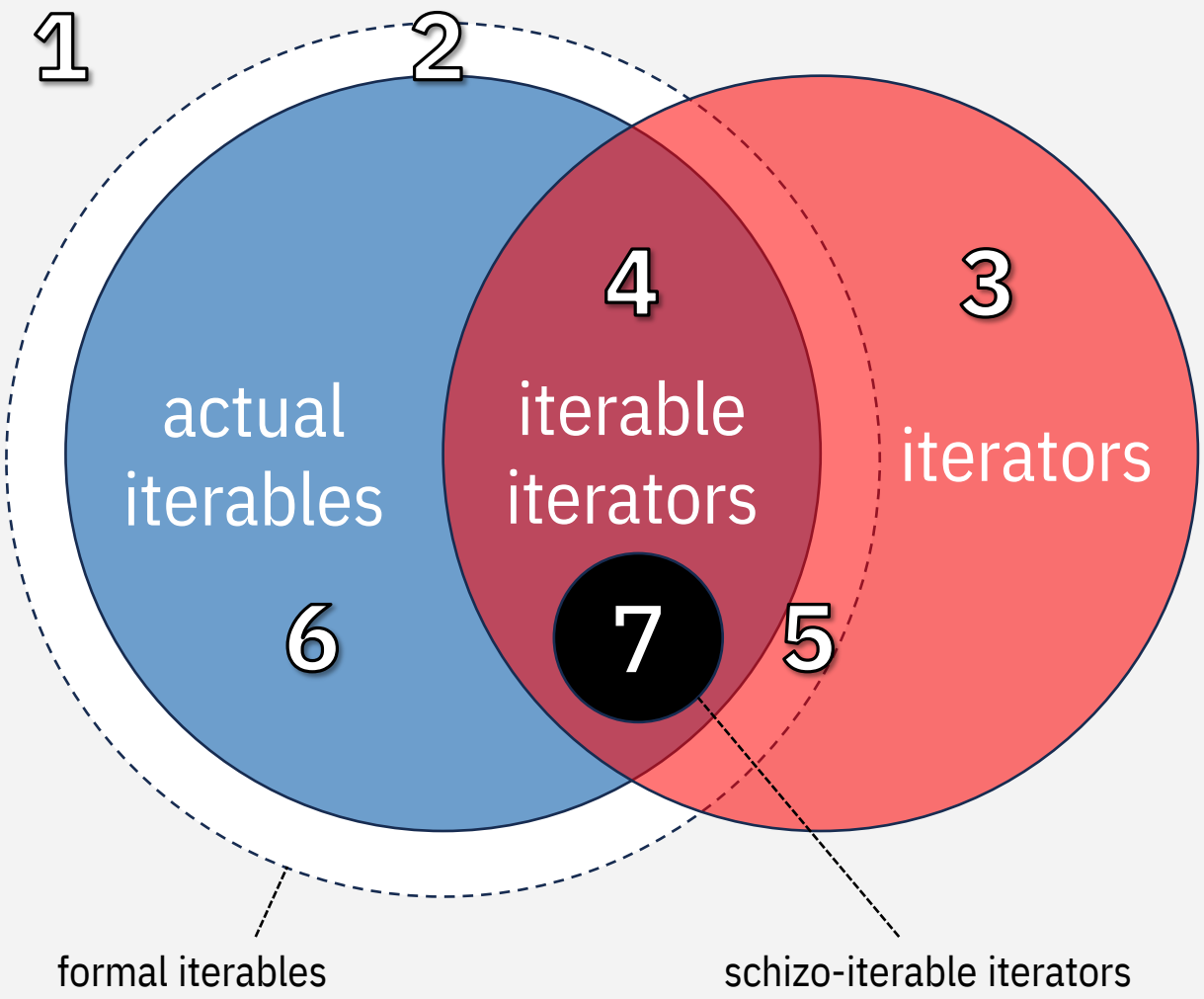
Te gebruiken als argument voor **iter**-functie:



Te gebruiken als argument voor **next**-functie:



Eulerdiagram: **Zeven bijbehorende voorbeelden**



Te gebruiken achter **in** in **for**-statement:



Te gebruiken als argument voor **iter**-functie:

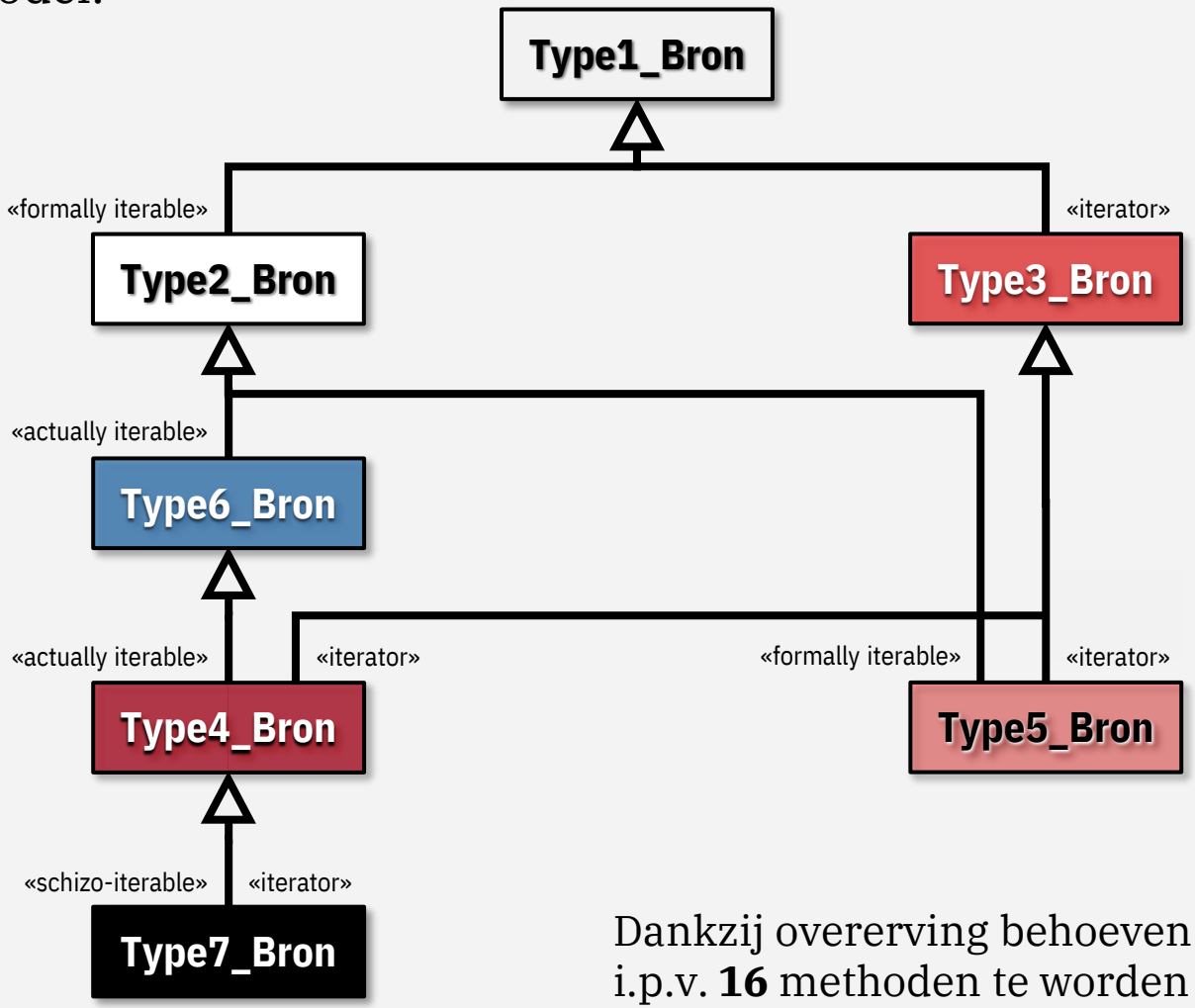


Te gebruiken als argument voor **next**-functie:



Klassendiagram: **Zeven representatieve classes**

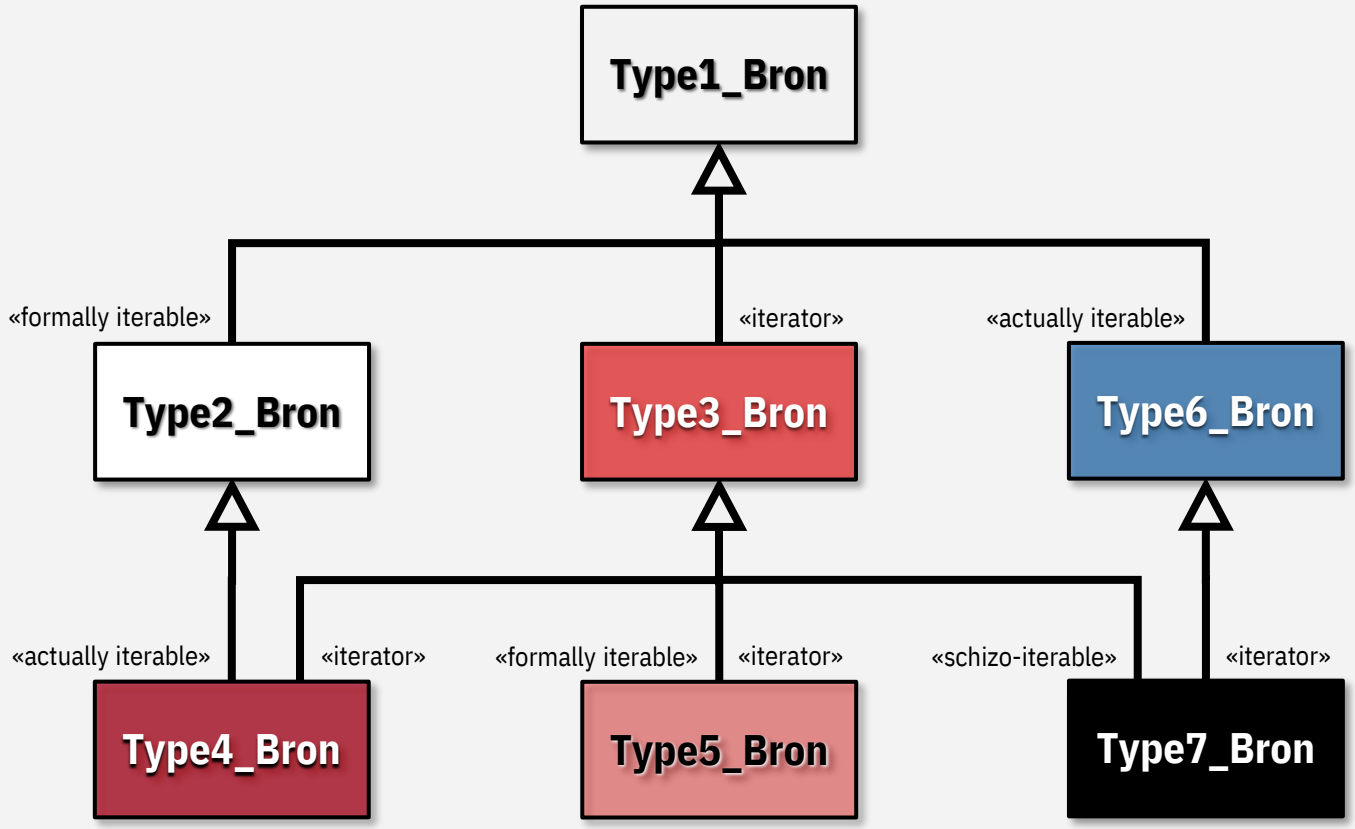
Logisch model:



Dankzij overerving behoeven er slechts **5** i.p.v. **16** methoden te worden gedefinieerd.

Klassendiagram: **Zeven representatieve classes**

Technische implementatie:



Ook in deze uitwerking behoeven er slechts **5** methoden te worden gedefinieerd.

Tabel: **Zeven representatieve classes**

#	@	Class	Kenschets	Methoden		
1		**Type1_Bron**	Object	`__init__`		
2		**Type2_Bron**	Pseudo-iterabel object	`__init__`	`__iter__`	
3		**Type3_Bron**	Iterator	`__init__`		`__next__`
4		**Type4_Bron**	Iterabele iterator	`__init__`	`__iter__`	`__next__`
5		**Type5_Bron**	Pseudo-iterabele iterator	`__init__`	`__iter__`	`__next__`
6		**Type6_Bron**	Iterabel object	`__init__`	`__iter__`	
7		**Type7_Bron**	Schizo-iterabele iterator	`__init__`	`__iter__`	`__next__`

<methode> = geërfde methode

<methode> = doet niet wat er van verwacht wordt

<methode> = doet wat er van verwacht wordt

<methode> = doet wat er van verwacht wordt, maar is overtollig

Testfunctie: **itereer_over(**<*bron*>**)**

```
def itereer_over(bron):
    print("\n-----", bron.naam, "-----")

    try:  # is dit object iterabel?
        print("iter     :", iter(bron))
    except Exception as e:
        print("iter     :", e)

    try:  # is dit object een iterator?
        print("next     :", next(bron))
    except Exception as e:
        print("next     :", e)

    try:  # bestaat er enige iterator voor dit object?
        print("next/iter:", next(iter(bron)))
    except Exception as e:
        print("next/iter:", e)

    print("for      :", end=" ")
    try:  # verdere iteratie over dit object mogelijk?
        for pakketje in bron:
            print(pakketje, end=" ")
        print()
    except Exception as e:
        print(e)
```

Dit zou een iterator moeten opleveren

Dit zou één iteratie moeten uitvoeren...

...en anders zou dit wel één iteratie moeten uitvoeren

Dit zou het itereren moeten voortzetten tot het eind

1 Niet iterabel, geen iterator



```
#---- Niet iterabel, geen iterator ----#
class Type1_Bron:
    def __init__(self, naam, lijst):
      self.naam  = naam
      self.lijst = lijst


bron1 = Type1_Bron("bron1", [1, 2, 3, 4])
itereer_over(bron1)
```

```
----- bron1 -----
iter     : 'Type1_Bron' object is not iterable
next     : 'Type1_Bron' object is not an iterator
next/iter: 'Type1_Bron' object is not iterable
for      : 'Type1_Bron' object is not iterable
```

2 Pseudo-iterabel, geen iterator



```
#---- Formeel iterabel, feitelijk niet, geen iterator ----#
class Type2_Bron(Type1_Bron):
    def __iter__(self):
        return self


bron2 = Type2_Bron("bron2", [1, 2, 3, 4])
itereer_over(bron2)
```

In deze context geen iterator, want **__next__** ontbreekt

```
----- bron2 -----
iter     : iter() returned non-iterator of type 'Type2_Bron'
next     : 'Type2_Bron' object is not an iterator
next/iter: iter() returned non-iterator of type 'Type2_Bron'
for      : iter() returned non-iterator of type 'Type2_Bron'
```

3 Niet iterabel, wel een iterator



```
#---- Niet iterabel, wel een iterator ----#
class Type3_Bron(Type1_Bron):
    def __next__(self):
        if len(self.lijst) > 0:
            return self.lijst.pop(0)
        raise StopIteration()

bron3 = Type3_Bron("bron3", [1, 2, 3, 4])
itereer_over(bron3)
```

```
----- bron3 -----
iter     : 'Type3_Bron' object is not iterable
next     : 1
next/iter: 'Type3_Bron' object is not iterable
for      : 'Type3_Bron' object is not iterable
```

4 Feitelijk iterabel, tevens een iterator



```
#---- Feitelijk iterabel, tevens een iterator ----#
class Type4_Bron(Type2_Bron, Type3_Bron):
    pass

bron4 = Type4_Bron("bron4", [1, 2, 3, 4])
itereer_over(bron4)
```

Erft **__iter__** (die **self** retourneert, in deze context een iterator) van Type2_Bron

Erft **__next__** van Type3_Bron

```
----- bron4 -----
iter     : <__main__.Type4_Bron object at 0x000002C3E92F0DD0>
next     : 1
next/iter: 2
for      : 3 4
```

Steeds hetzelfde object!

5 Pseudo-iterabel, wel een iterator



```
#---- Formeel iterabel, feitelijk niet, wel een iterator ----#
class Type5_Bron(Type3_Bron):
    def __iter__(self):
        return self.lijst


bron5 = Type5_Bron("bron5", [1, 2, 3, 4])
itereer_over(bron5)
```

Erft **__next__** van Type3_Bron

Een list is geen iterator

```
----- bron5 -----
iter     : iter() returned non-iterator of type 'list'
next     : 1
next/iter: iter() returned non-iterator of type 'list'
for      : iter() returned non-iterator of type 'list'
```

6 Feitelijk iterabel, externe iterator



```
#---- Feitelijk iterabel, maar
#     afgifte overgedragen aan externe iterator ----#
class Type6_Bron(Type1_Bron):
    def __iter__(self):
        return Type4_Bron("anoniem“,
                          [10*x for x in self.lijst])

bron6 = Type6_Bron("bron6", [1, 2, 3, 4])
itereer_over(bron6)
```

```
----- bron6 -----
iter     : <__main__.Type4_Bron object at 0x000002C3E92F1490>
next     : 'Type6_Bron' object is not an iterator
next/iter: 10
for      : 10 20 30 40
```

Nieuw object!

7 Iterabel, interne + externe iterator



Schizo-

```
#---- Feitelijk een iterabele iterator, maar
#     afgifte toch overgedragen aan externe iterator ----#
class Type7_Bron(Type3_Bron, Type6_Bron):
    pass


bron7 = Type7_Bron("bron7", [1, 2, 3, 4])
itereer_over(bron7)
```

Erft **__next__** van Type3_Bron

Erft **__iter__** van Type6_Bron

```
----- bron7 -----
iter     : <__main__.Type4_Bron object at 0x000002C3E92F1490>
next     : 1
next/iter: 20
for      : 20 30 40
```

Literatuur

- Pieter Spronck: *De programmeursleerling – Leren coderen met Python 3*
  versie 1.1.1, 2024
  (Nederlandse vertaling van *The Coder's Apprentice*).

- Meindert Meindertsma: Python-programma “iteriolen.py”)
  21 januari 2024
  (de integrale achterliggende code van deze presentatie).

- ____: *Itereren met while & for*
  18 en 22 januari 2024 (bijgewerkt op 11 november 2026)
  (PDF “while-and-for-2026.pdf” van een PowerPoint-presentatie).
  In dit verband zijn de slides 7–10 en 13–16 het meest interessant.

- ____: *Een boom van iteriolen in Python*
  11 november 2026
  (PDF “iteriolenboom.pdf” van een PowerPoint-presentatie).

- Kritische reacties op mijn producten zijn welkom (bijvoorbeeld via mijn e-mailadres **ledepot2023@gmail.com**).