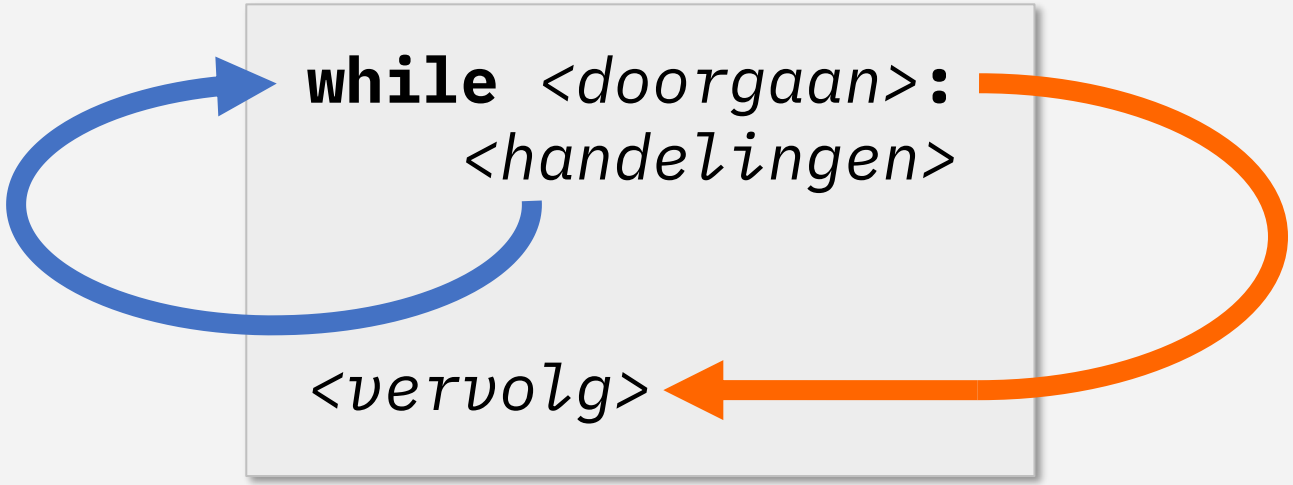
GEACTUALISEERD
11 NOV 2026



ITEREREN MET **while** & **for**

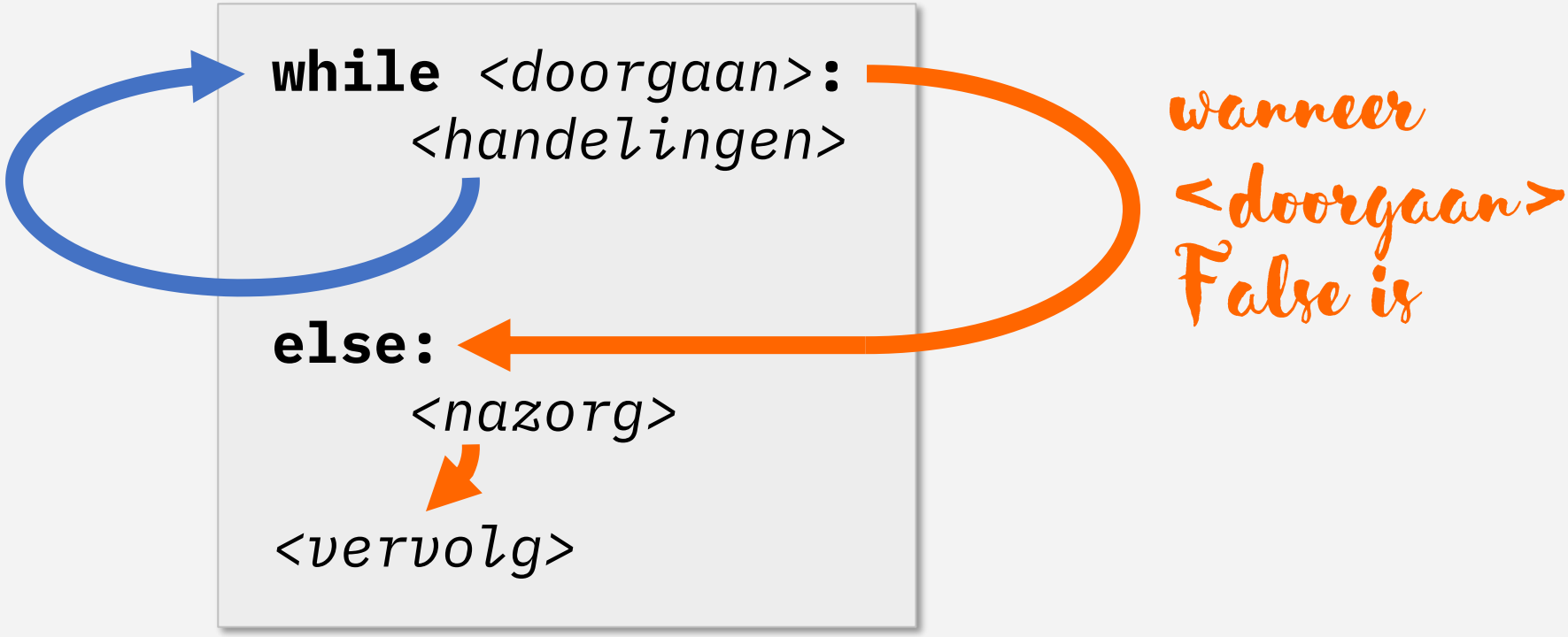
Meindert Meindertsma — 18 en 22 januari 2024

Hoofdstuk 7: **while ...**

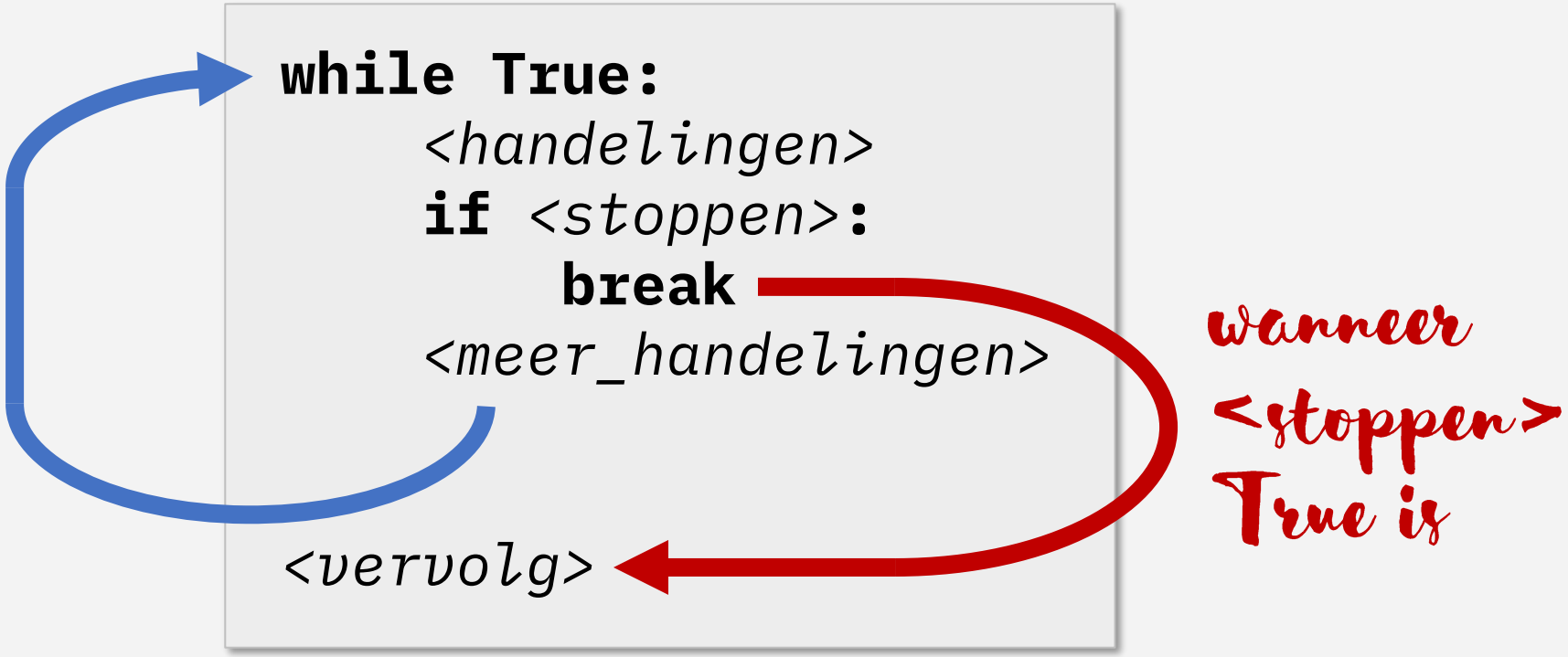


wanneer <doorgaan> False is

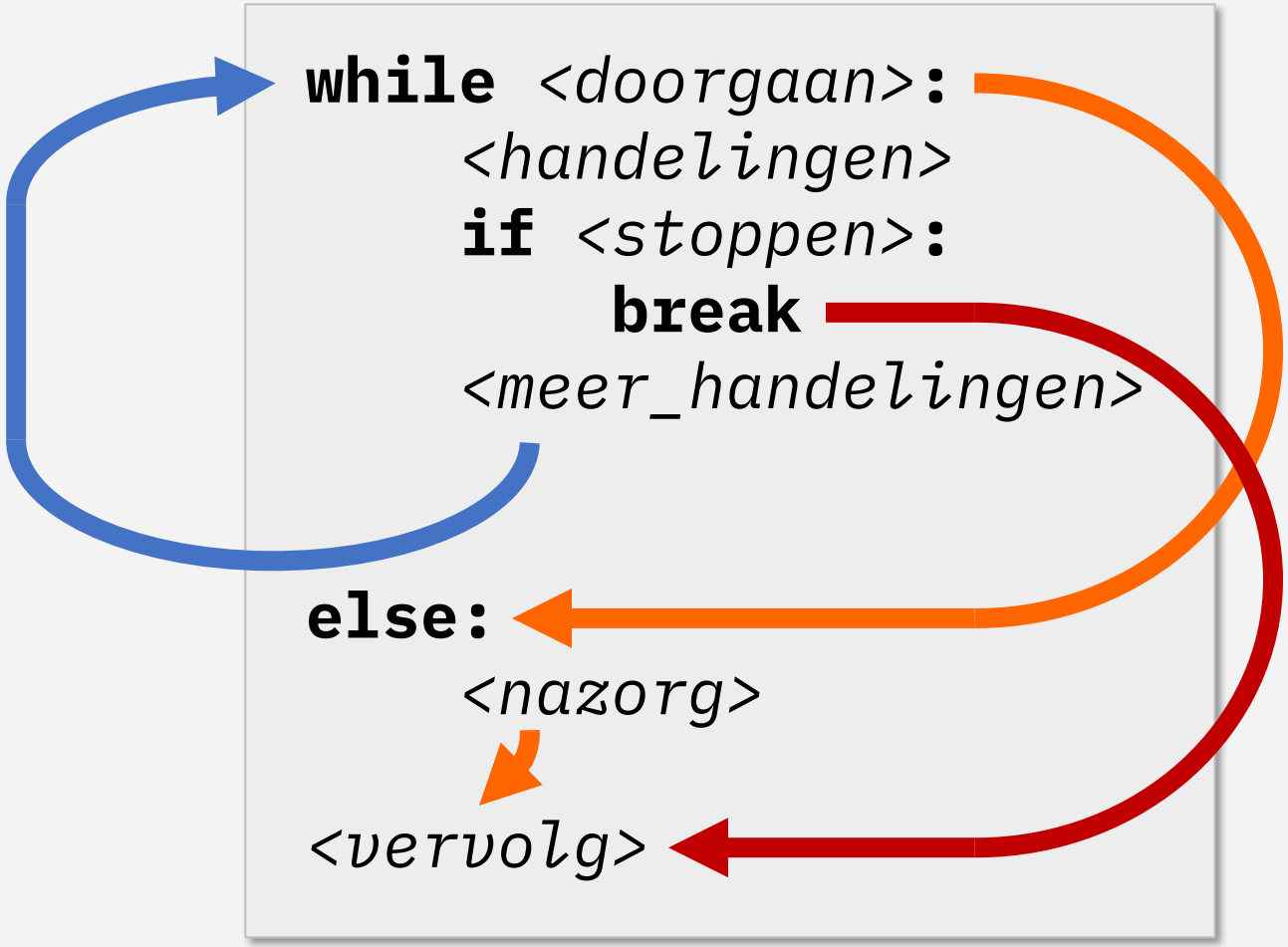
Hoofdstuk 7: **while … else …**



Hoofdstuk 7: **while … break …**



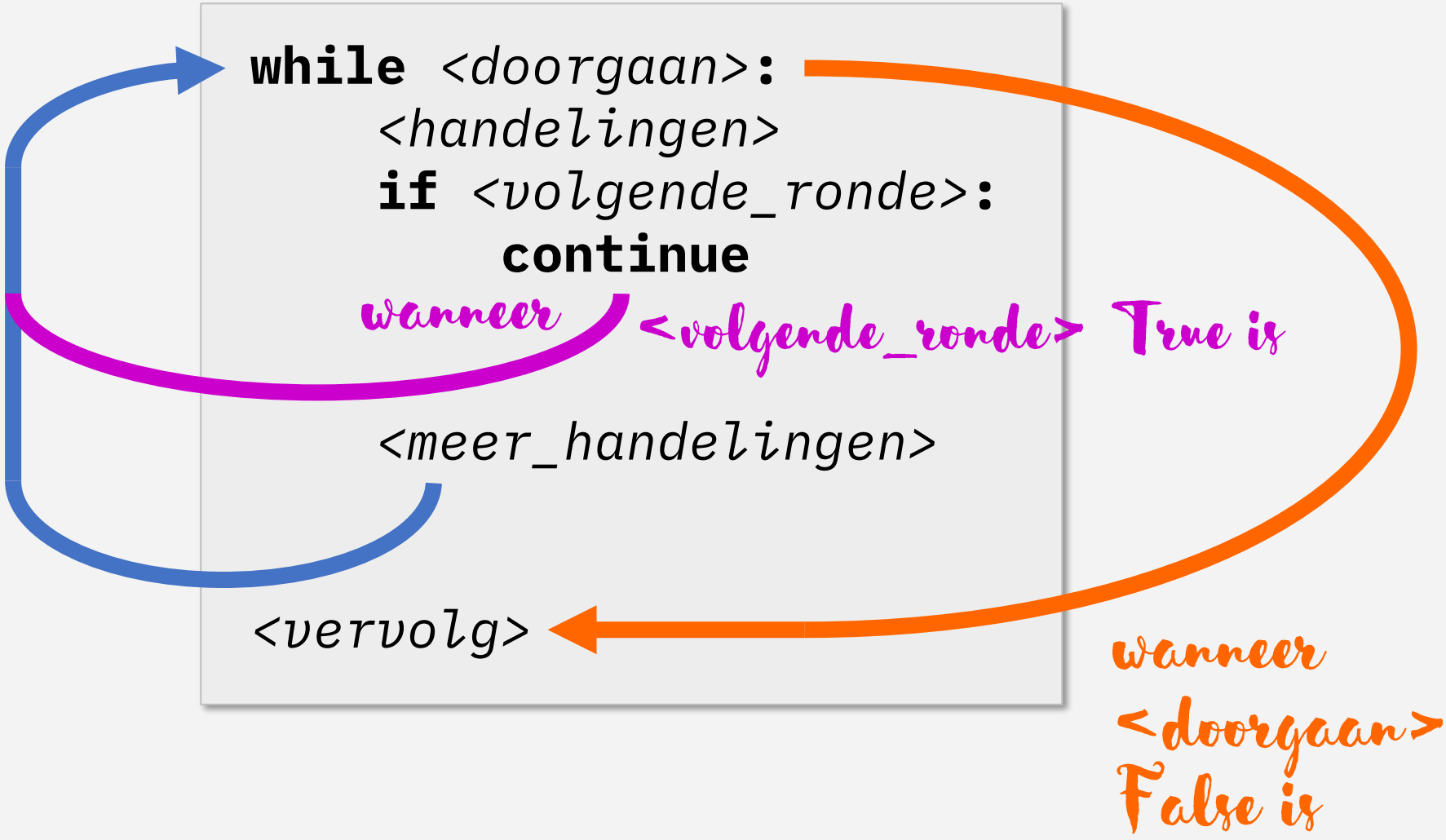
Hoofdstuk 7: **while … break … else …**



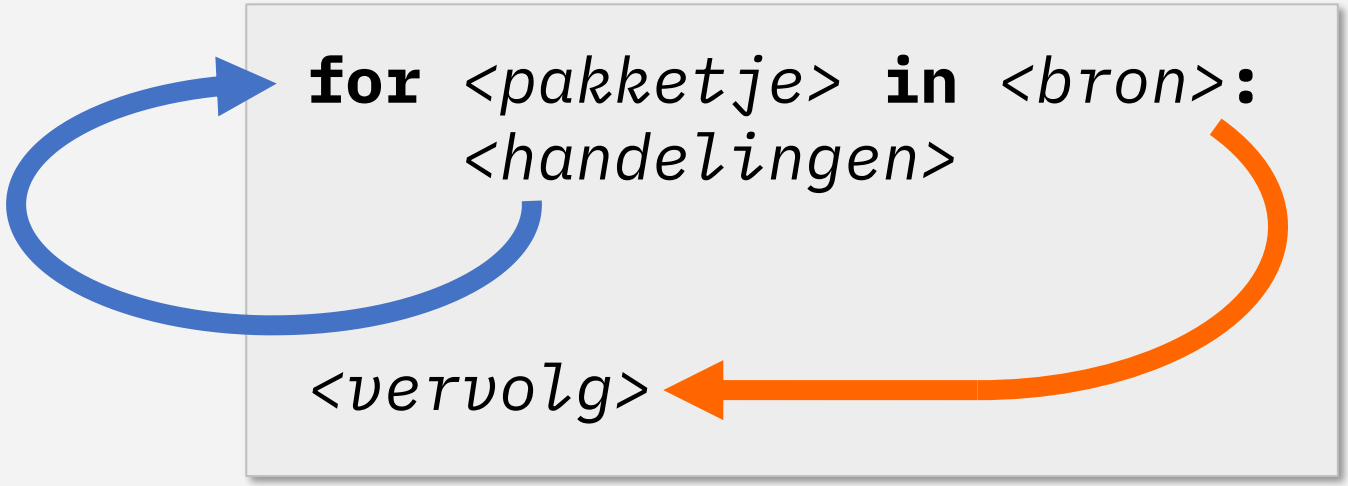
wanneer <doorgaan> False is

wanneer <stoppen> True is

Hoofdstuk 7: **while … continue …**



Hoofdstuk 7: **for …**



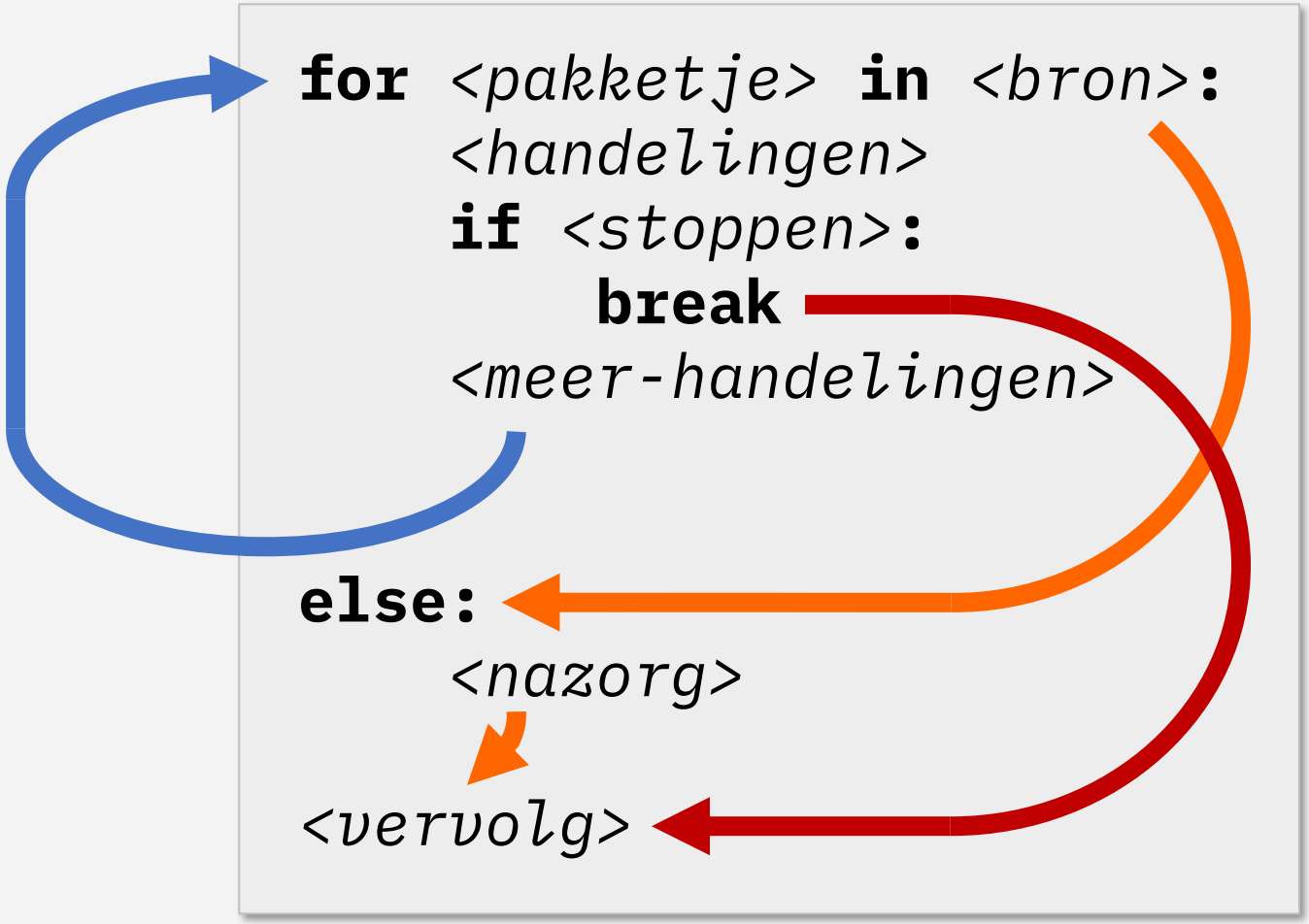
Het object *<bron>* representeert:

- een aanwezige voorraad van ‘pakketjes’, of
- een maker of verwerker van ‘pakketjes’ die op afroep zullen worden geproduceerd dan wel op andere wijze beschikbaar gesteld.

Technisch bezien:

- bestaat een *<pakketje>* uit een of meer iteratievariabelen;
- is *<bron>* een **‘iterabel’** object (zie slide 10).

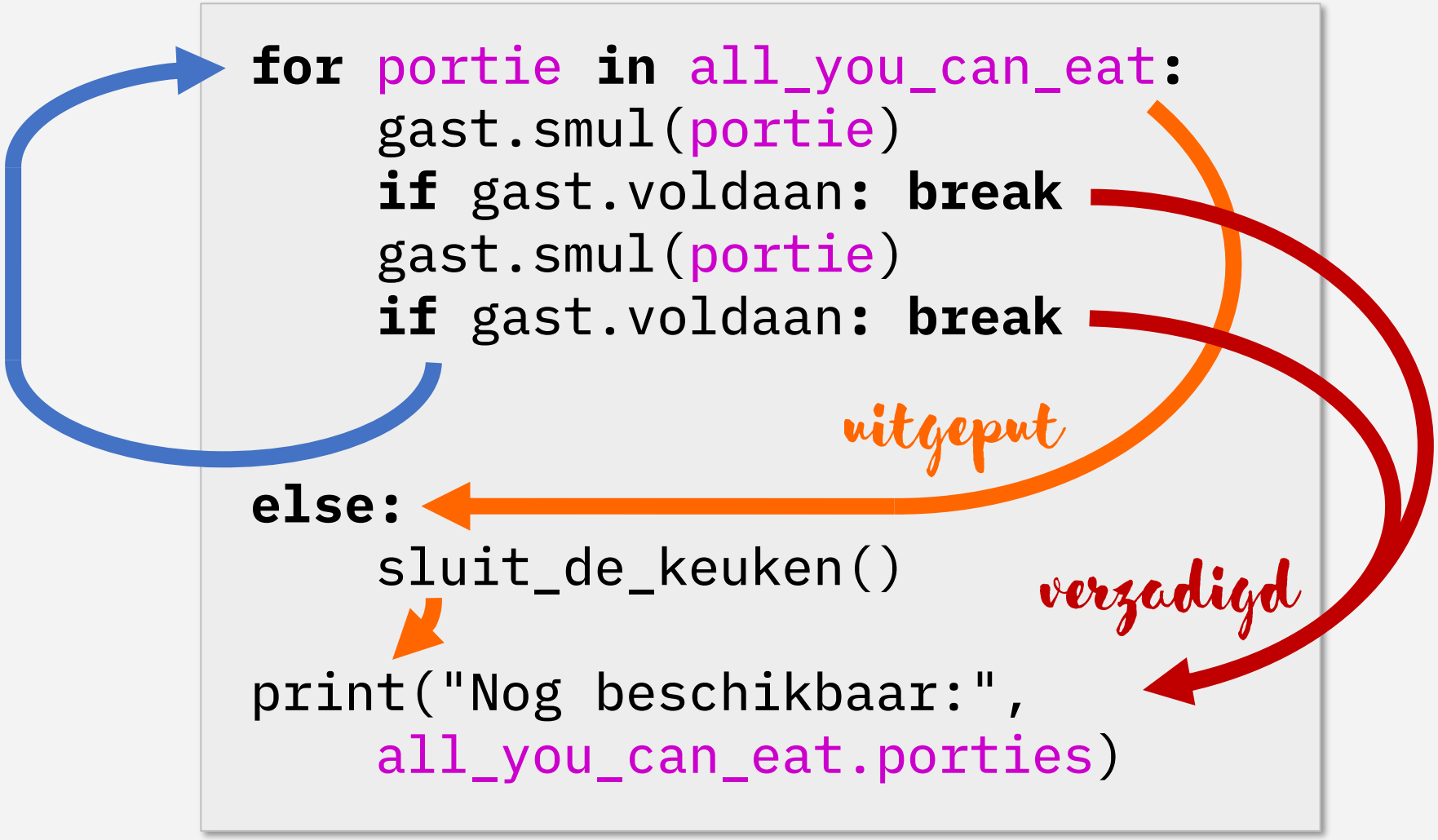
Hoofdstuk 7: **for … break … else …**



wanneer er niets meer leverbaar is

wanneer <stoppen> True is

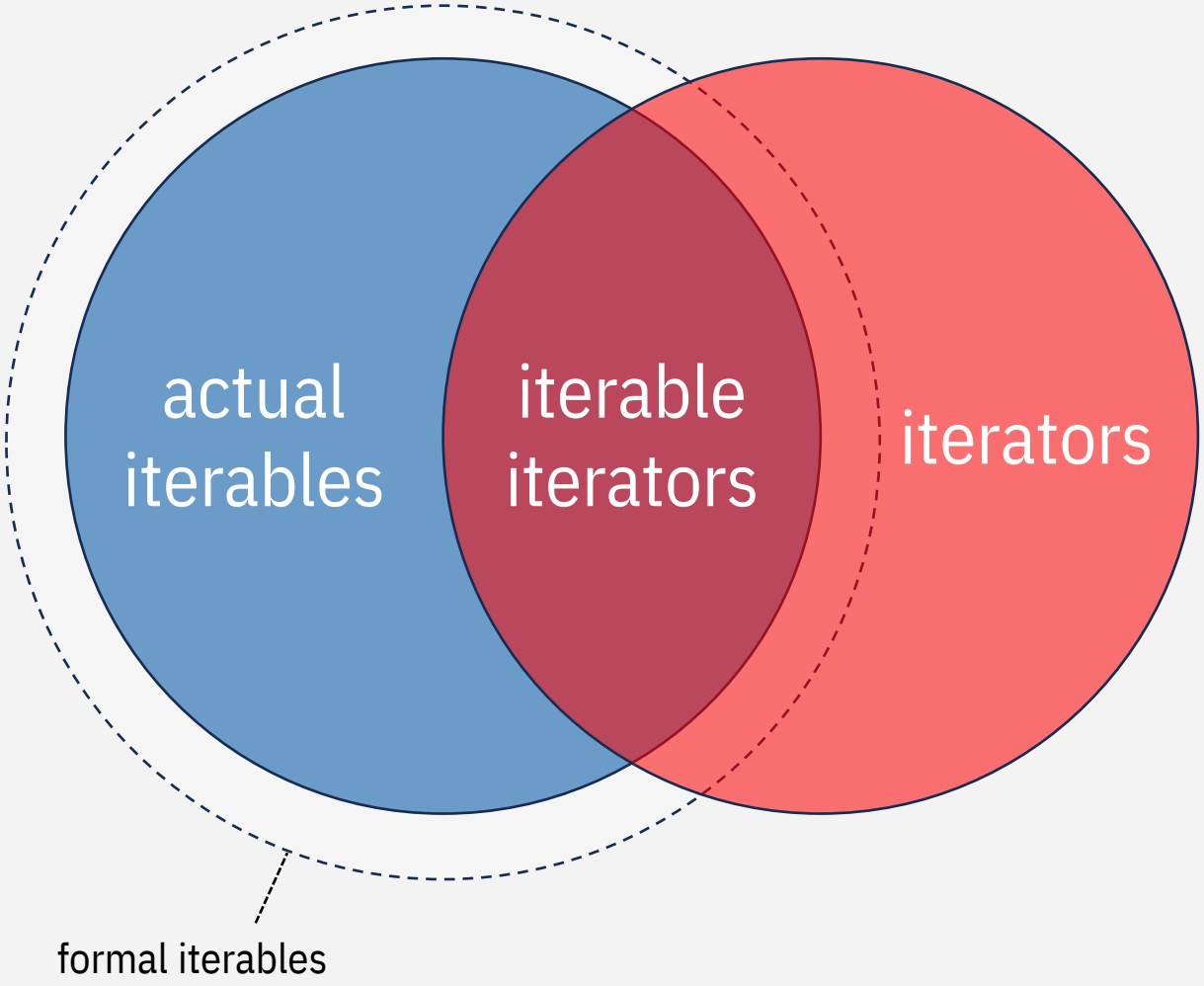
Hoofdstukken 7 en 23: **All You Can Eat**



Hoofdstuk 23: de begrippen **‘iterable’** en **‘iterator’**

- Een object is *formeel* **iterabel** indien het een methode **`__iter__`** bevat.
  Zo niet (zie slide 14), dan geeft **`iter(`***`<bron>`***`)`** de foutmelding:
  “TypeError: …… object is not iterable”
- Een object is *feitelijk* **iterabel** indien zijn methode **`__iter__`** een **iterator** retourneert.
  Zo niet (zie slide 14), dan geeft **`iter(`***`<bron>`***`)`** de foutmelding:
  “TypeError: iter() returned non-iterator of type ……”
- Een object is een **iterator** indien het een methode **`__next__`** bevat.
  Zo niet (zie slide 15), dan geeft **`next(`***`<verdeler>`***`)`** de foutmelding:
  “TypeError: …… object is not an iterator”
- Een **iterabel** object kan tegelijkertijd een **iterator** zijn. Noodzakelijk is dit niet. Wanneer het wel het geval is, zijn deze configuraties mogelijk:
  - Het object is zijn eigen iterator, oftewel zijn methode **`__iter__`** retourneert **`self`**.
  - Het object is *niet* zijn eigen iterator, want zijn methode **`__iter__`** retourneert, ook al heeft het zelf een eigen **`__next__`**-methode, toch een ander object.

Hoofdstuk 23: de begrippen **‘iterable’** en **‘iterator’**



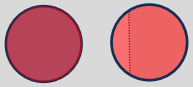
Te gebruiken achter **in** in **for**-statement:



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



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



Hoofdstuk 23: de ingebouwde functies **iter** en **next**

Eerst het hoofdstuk
maar eens helemaal
van begin tot aan
de eindopgaven
doorploegen…

Hoofdstuk 23: wat **for … in** <*bron*> doet

1. De methode <*bron*>**.__iter__** aanroepen om de iterator <*verdeler*> van <*bron*> op te halen. Daarbij kan <*verdeler*> hetzelfde object blijken als <*bron*>, maar dat doet er verder niet toe.
2. Heeft **for** geen methode **__iter__** in <*bron*> aangetroffen, dan heeft het een **TypeError** gegenereerd en houdt het voor **for** (en voor het hele programma, als die fout nergens wordt afgevangen) op. Is alles echter in orde, dan door naar stap 3.
3. De methode <*verdeler*>**.__next__** aanroepen.
4. Wanneer **__next__()** een **StopIteration** heeft gegenereerd, deze afvangen en inderdaad maar gedwee stoppen met itereren. Wel de eventuele **else**-tak nog even uitvoeren.
5. Anders doorgaan met de van **__next__()** ontvangen waarden voor de iteratievariabelen. Dat wil zeggen, de code in het **for**-blok uitvoeren.
6. Terug naar stap 3, tenzij het **for**-blok een **break** forceert of er een fout optreedt.

Hoofdstuk 23: wat **iter(**<*bron*>**)** doet

1. Kijken of <*bron*> een methode **__iter__** bevat. Indien niet en ook geen methode **__getitem__***), dan error:
   "TypeError: …… object is not iterable"
2. De methode <*bron*>**.__iter__** aanroepen. Deze retourneert een object <*verdeler*> (dit kan wel of niet hetzelfde object als <*bron*> zijn).
3. Kijken of <*verdeler*> een methode **__next__** bevat. Zo niet, dan error:
   "TypeError: iter() returned non-iterator of type ……"
4. De **iterator** <*verdeler*> retourneren.

Vervolgens kan de functie **next** met <*verdeler*> aan de slag of kan de methode <*verdeler*>**.__next__** rechtstreeks worden aangeroepen.

---

*) De iteratievoorzieningen rond de methode **__getitem__** zijn een relict van Python 2. De methode wordt pas aangesproken als **__iter__** niet bestaat. Spronck behandelt de mogelijke rol bij iteraties via **for** niet, en ik doe dat hier evenmin.

Hoofdstuk 23: wat **next(***<verdeler>[, <default>]***)** doet

1. Nagaan of *<verdeler>* een methode **__next__** bevat. Zo niet, dan error: "TypeError: …… object is not an iterator"
2. De methode *<verdeler>*.**__next__** aanroepen om een pakket van waarden uit dat object op te halen.
3. Deze waarden als returnwaarden doorgeven.
4. Kan **__next__()** niets meer leveren en genereert het daarom de exception **StopIteration**, dan hangt het ervan af of **next** een tweede argument *<default>* meegekregen heeft :
   a. Zo ja, dan de exception afvangen en de waarde *<default>* retourneren.
   b. Zo nee, dan de uitzonderingstoestand **StopIteration** aanhouden en zo eindigen.

Literatuur

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

- Meindert Meindertsma
  Een jaar of wat – Tijdelijke uitstalling van weetjes en ideetjes
  - Ingebouwde 'functies' in Python
    Een inventarisatie van Python-functies, waarin onder meer is aangegeven welke van die functies een iterator genereren (gemerkt met ↻) en welke constructors een iterabel object instantiëren (gemerkt met »).
  - Iteratiemechanismen
    Een bundel van artikeltjes over itereren, voornamelijk voor zover het Python, Ruby en Lua betreft. Huidige staat: nog lang niet af, en wat al geschreven is, is voor verbetering vatbaar. Geen stof voor beginnende programmeurs. Kritische reacties zijn welkom (bijvoorbeeld via mijn e-mailadres **ledepot2023@ymail.com**).