Universiteit Gent
Faculteit Wetenschappen

Programmeren in Python

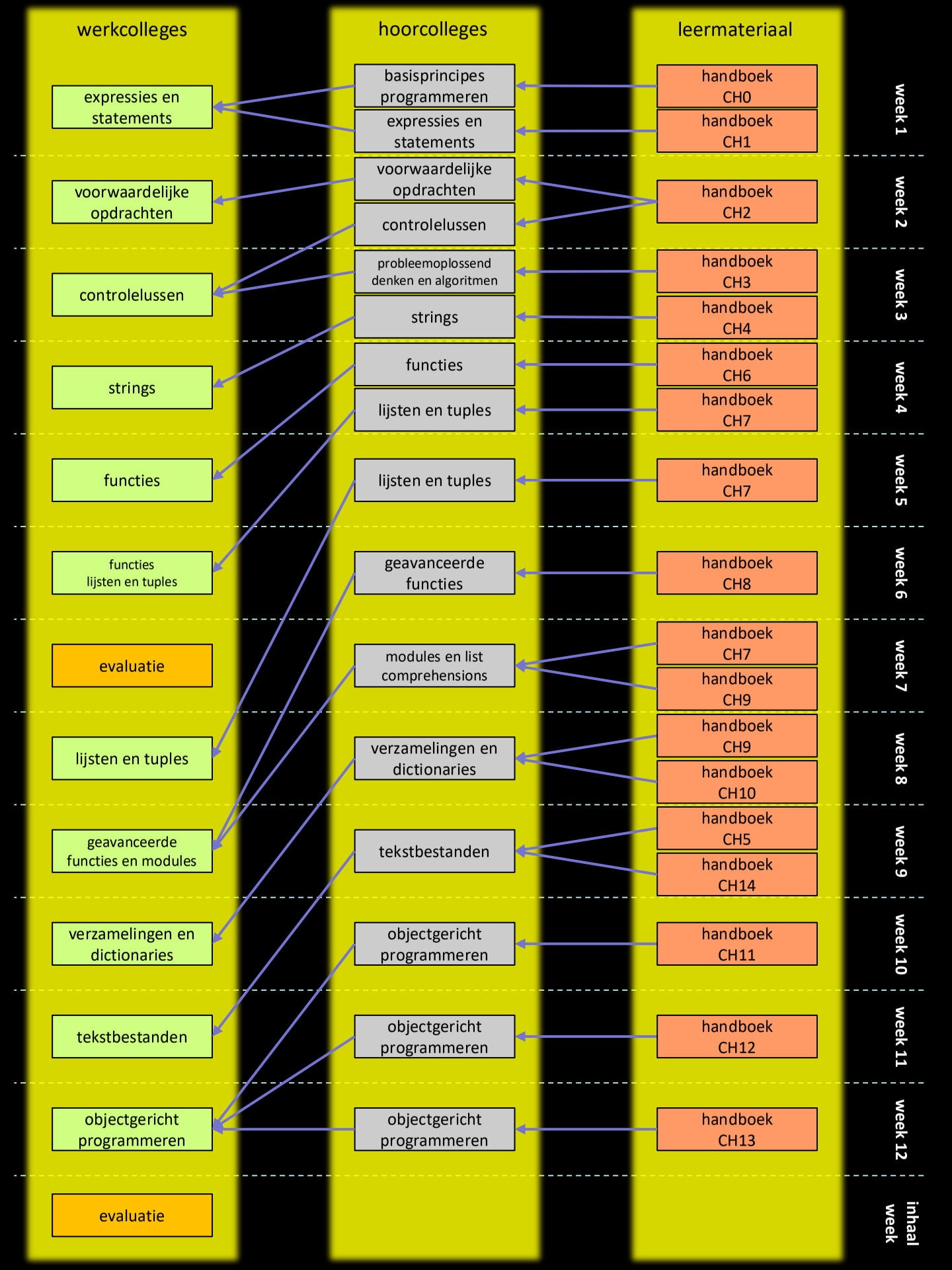
modules en list comprehensions

prof. dr. Peter Dawyndt

peter.dawyndt@ugent.be

@dawyndt

UNIVERSITEIT GENT


werkcolleges
hoorcolleges
leermateriaal
expressies en statements
voorwaardelijke opdrachten
controlelussen
strings
functies
functies lijsten en tuples
evaluatie
lijsten en tuples
geavanceerde functies en modules
verzamelingen en dictionaries
tekstbestanden
objectgericht programmeren
evaluatie
basisprincipes programmeren
expressies en statements
voorwaardelijke opdrachten
controlelussen
probleemoplossend denken en algoritmen
strings
functies
lijsten en tuples
lijsten en tuples
geavanceerde functies
modules en list comprehensions
verzamelingen en dictionaries
tekstbestanden
objectgericht programmeren
objectgericht programmeren
objectgericht programmeren
handboek CH0
handboek CH1
handboek CH2
handboek CH3
handboek CH4
handboek CH6
handboek CH7
handboek CH7
handboek CH8
handboek CH7
handboek CH9
handboek CH9
handboek CH10
handboek CH5
handboek CH14
handboek CH11
handboek CH12
handboek CH13
week 1
week 2
week 3
week 4
week 5
week 6
week 7
week 8
week 9
week 10
week 11
week 12
inhaal week

Modules

Modules

- **module**: bestand met Python definities en statements
  - doel: definities hergebruiken in andere Python code
  - **The Python Standard Library**
    - modulebibliotheek die standaard meegeleverd wordt
    - "batteries included"
  - Python + standard library + wetenschappelijke modules
    - Enthought Python Distribution (EDP)
    - Anaconda
  - nog meer modules …
    - Python Package Index (aka "The Cheese Shop")

"Don't reinvent the wheel."

UNIVERSITEIT GENT

python™

ANACONDA

ENTHOUGHT SCIENTIFIC COMPUTING SOLUTIONS

BATTERIES INCLUDED

Modules aanmaken

- elk Python bestand is automatisch ook een module
  - toegankelijk maken via **`import`** statement
  - module is zelf ook een object met eigenschappen en methoden
  - eigenschappen en methoden aanspreken met punt (*dot operator*)

cirkel.py

```
"Module met cirkelmaten."

import math

def omtrek(straal):
    "Omtrek van cirkel."
    return 2 * math.pi * straal

def oppervlakte(straal):
    "Oppervlakte van cirkel."
    return math.pi * straal**2
```

```
>>> straal = 2.87
>>> import cirkel
>>> type(cirkel)
<class 'module'>
>>> dir(cirkel)
['__builtins__', '__cached__', '__doc__',
 '__file__', '__name__', '__package__', 'math',
 'omtrek', 'oppervlakte']
>>> cirkel.omtrek(straal)
18.0327418316
>>> cirkel.oppervlakte(straal)
25.8769845284
```

Modules aanmaken

- elk Python bestand is automatisch ook een module
  - toegankelijk maken via **`import`** statement
  - module is zelf ook een object met eigenschappen en methoden
  - eigenschappen en methoden aanspreken met punt (*dot operator*)

cirkel.py

```
"Module met cirkelmaten."

import math

def omtrek(straal):
    "Omtrek van cirkel."
    return 2 * math.pi * straal

def oppervlakte(straal):
    "Oppervlakte van cirkel."
    return math.pi * straal**2
```

```
>>> import cirkel
>>> cirkel.__name__
'cirkel'
>>> cirkel.__doc__
'Module met cirkelmaten.'
>>> help(cirkel)
Help on module cirkel:

NAME
    cirkel - Module met cirkelmaten.

FUNCTIONS
    omtrek(straal)
        Omtrek van cirkel.
    oppervlakte(straal)
        Oppervlakte van cirkel.
```

docstring

Modules aanmaken

- elk Python bestand is automatisch ook een module
  - toegankelijk maken via **`import`** statement
  - module is zelf ook een object met eigenschappen en methoden
  - eigenschappen en methoden aanspreken met punt (*dot operator*)

cirkel.py

```
"Module met cirkelmaten."

import math

def omtrek(straal):
    "Omtrek van cirkel."
    return 2 * math.pi * straal

def oppervlakte(straal):
    "Oppervlakte van cirkel."
    return math.pi * straal**2
```

analyse.py

```
import cirkel

straal = input('straal: ')

ftest = straal
if '.' in straal:  # float gegeven ?
    ftest = straal.replace('.', '', 1)

if ftest.isdigit():
    straal = float(straal)
    print(cirkel.omtrek(straal))
    print(cirkel.oppervlakte(straal))
```

Namespaces

- **namespace**: syntactische container die toelaat zelfde naam te gebruiken in modules, functies, methodes en klassen
- elke module heeft eigen namespace
  - identieke namen zorgen niet voor identificatieprobleem

module1.py

```
vraag = 'What is the meaning of life, the Universe, and everything?'
antwoord = 42
```

module2.py

```
vraag = 'What is your quest?'
antwoord = 'To seek the holy grail.'
```

```
>>> import module1
>>> import module2
>>> module1.vraag
'What is the meaning of life, the Universe, and everything?'
>>> module2.vraag
'What is your quest?'
```

UNIVERSITEIT GENT

Namespaces

- **namespace**: syntactische container die toelaat zelfde naam te gebruiken in modules, functies, methodes en klassen
- elke module heeft eigen namespace
  - identieke namen zorgen niet voor identificatieprobleem

module1.py

```
vraag = 'What is the meaning of life, the Universe, and everything?'
antwoord = 42
```

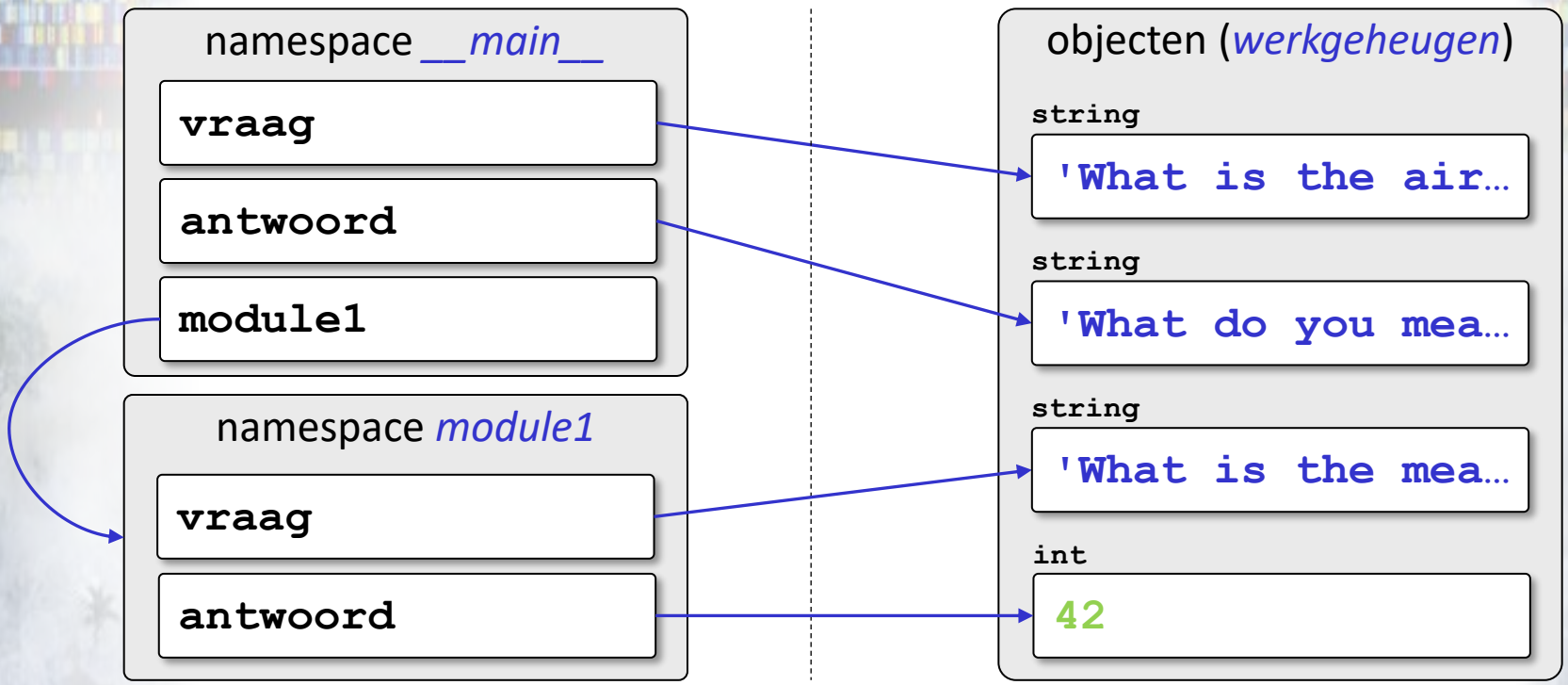
module2.py

```
vraag = 'What is your quest?'
antwoord = 'To seek the holy grail.'
```

```
>>> module1.antwoord
42
>>> module2.antwoord
'To seek the holy grail.'
```

UNIVERSITEIT GENT

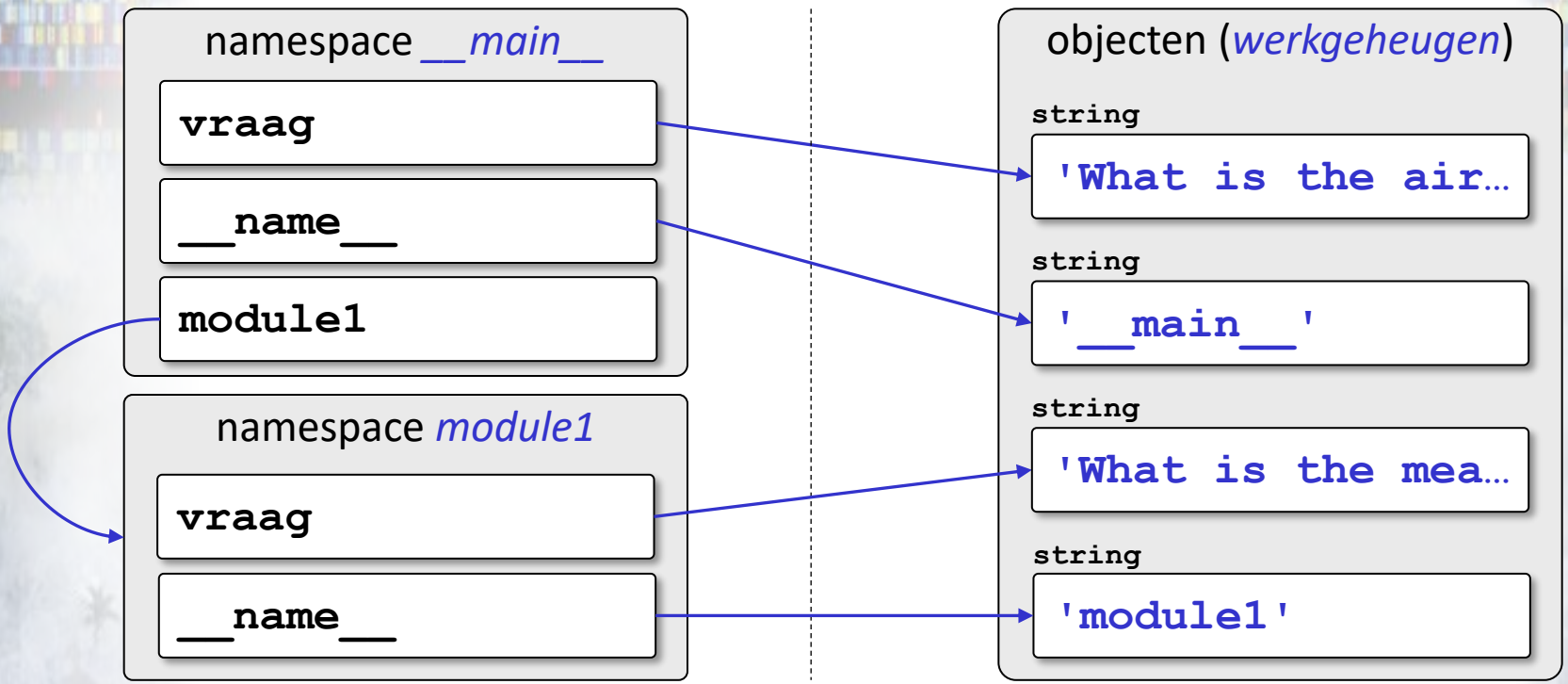
Namespaces




```
>>> vraag = 'What is the air-speed velocity of an unladen swallow?'
>>> antwoord = 'What do you mean? An African or European swallow?'
>>> import module1
>>> vraag
'What is the air-speed velocity of an unladen swallow?'
>>> module1.vraag
'What is the meaning of life, the Universe, and everything?'
```

UNIVERSITEIT GENT

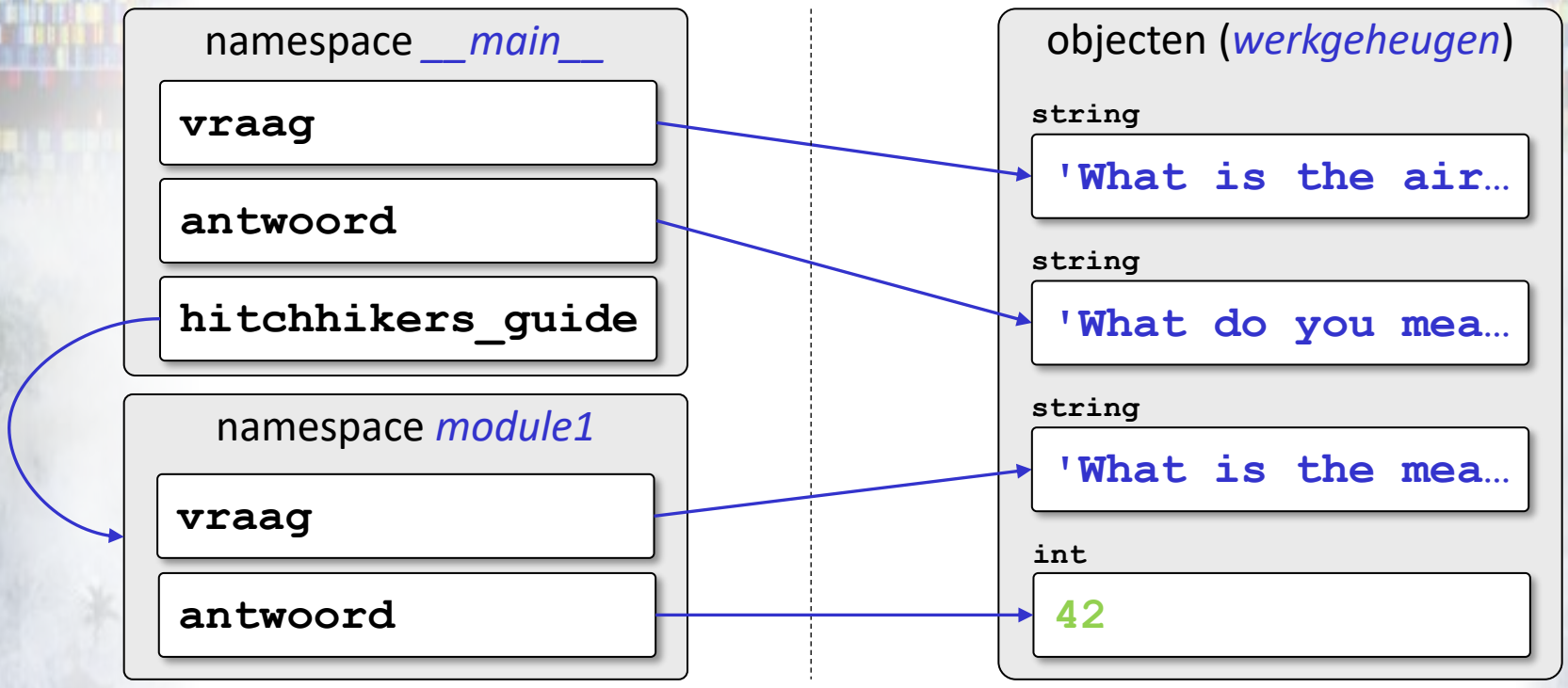
Namespaces




```
>>> import module1
>>> __name__
'__main__'
>>> module1.__name__
'module1'
```

UNIVERSITEIT GENT

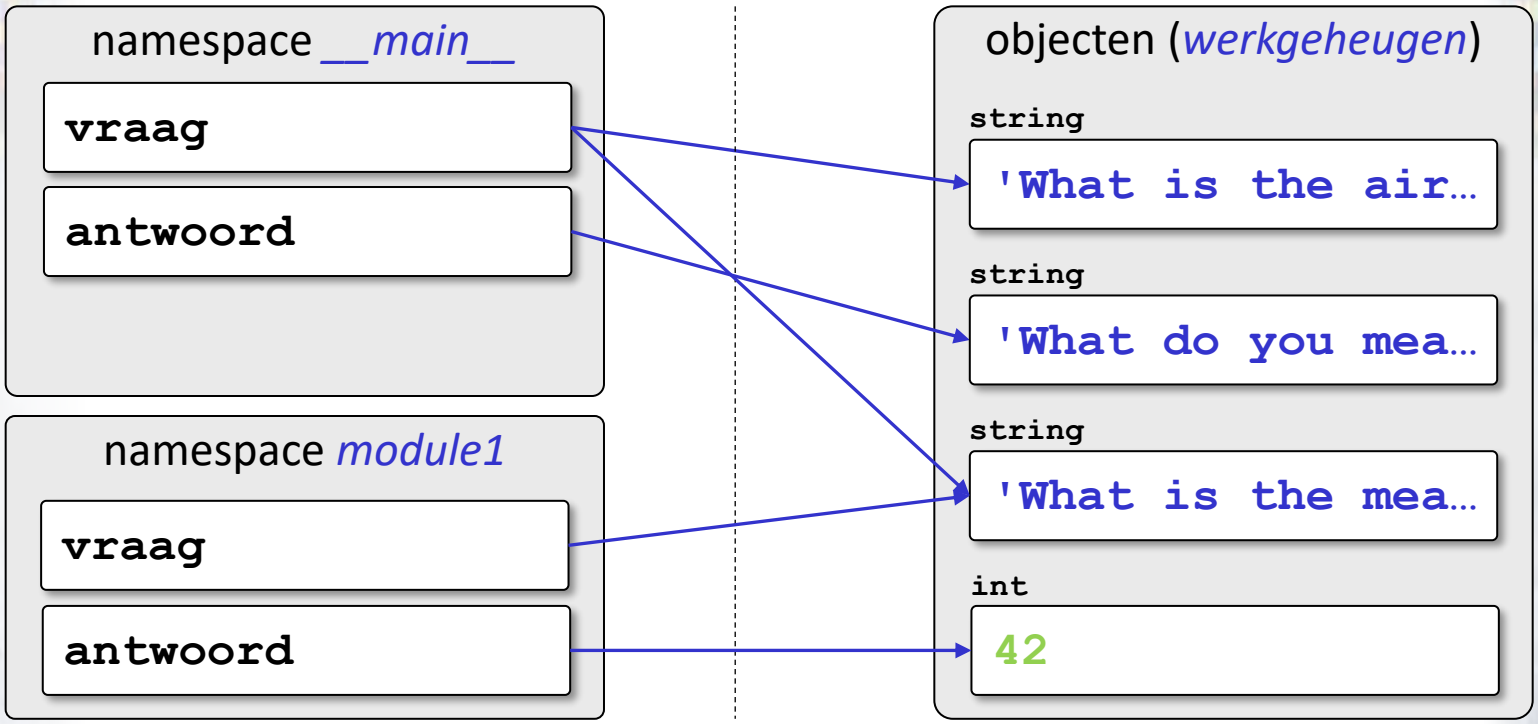
Alternatieve imports




```
>>> vraag = 'What is the air-speed velocity of an unladen swallow?'
>>> antwoord = 'What do you mean? An African or European swallow?'
>>> import module1 as hitchhikers_guide
>>> vraag
'What is the air-speed velocity of an unladen swallow?'
>>> hitchhikers_guide.vraag
'What is the meaning of life, the Universe, and everything?'
```

UNIVERSITEIT GENT

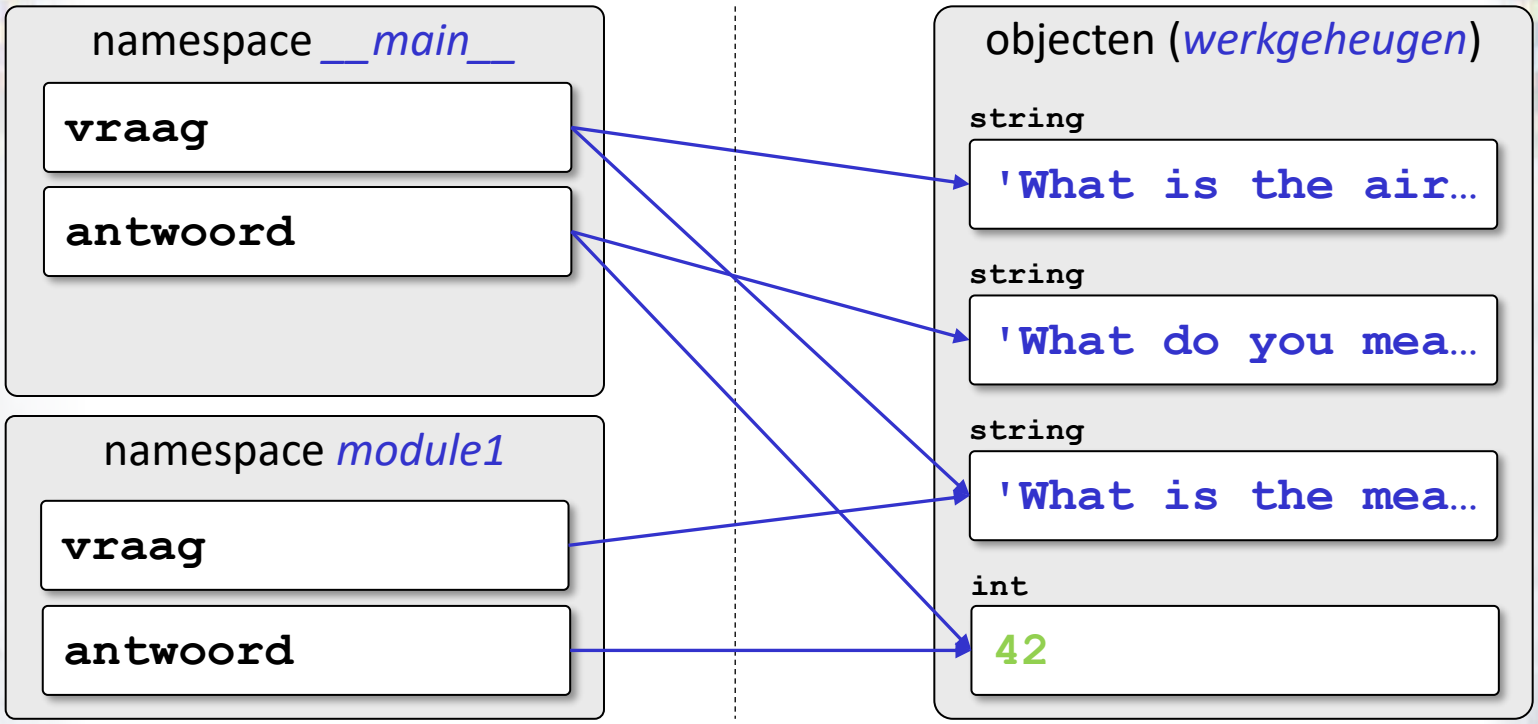
Alternatieve imports




```
>>> vraag = 'What is the air-speed velocity of an unladen swallow?'
>>> antwoord = 'What do you mean? An African or European swallow?'
>>> from module1 import vraag
>>> vraag
'What is the meaning of life, the Universe, and everything?'
>>> antwoord
'What do you mean? An African or European swallow?'
```

UNIVERSITEIT GENT

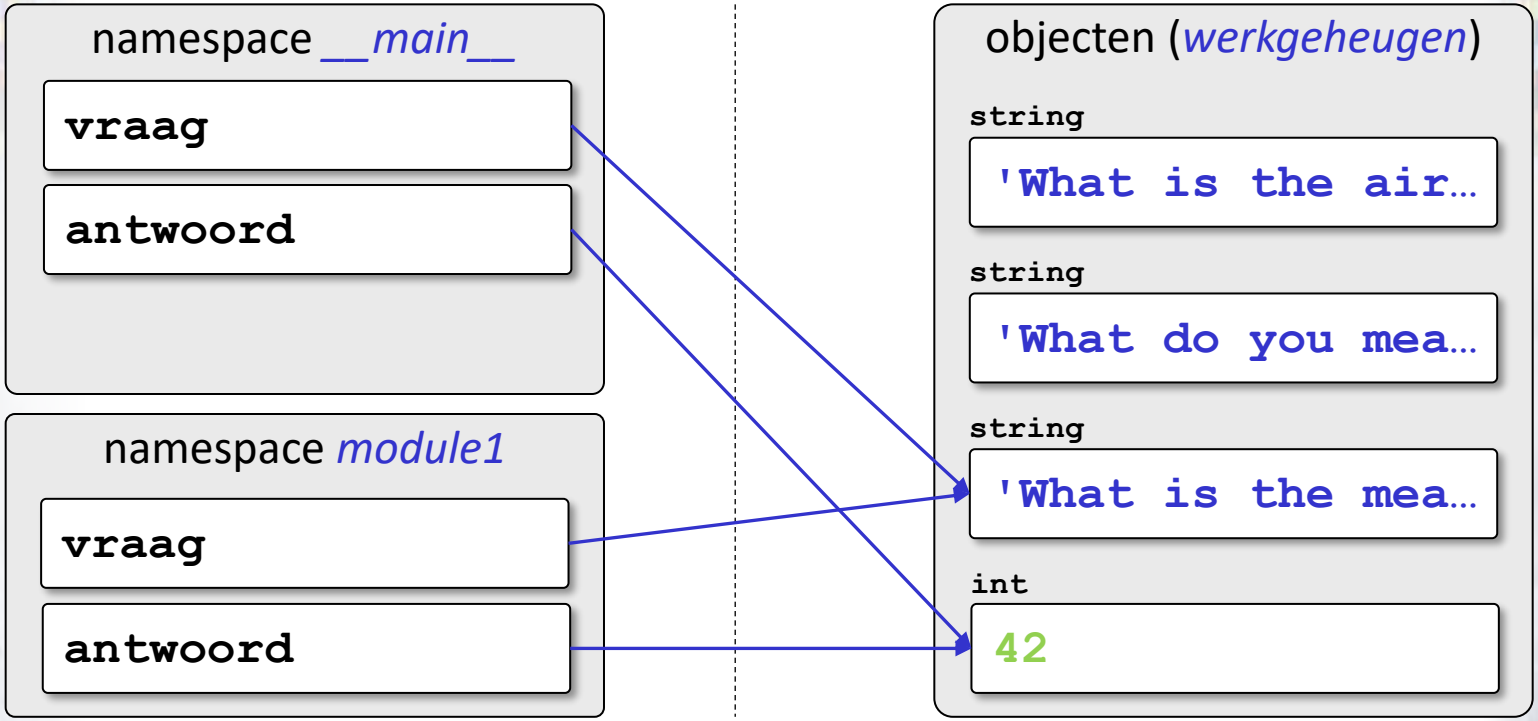
Alternatieve imports




```
>>> vraag = 'What is the air-speed velocity of an unladen swallow?'
>>> antwoord = 'What do you mean? An African or European swallow?'
>>> from module1 import vraag, antwoord
>>> vraag
'What is the meaning of life, the Universe, and everything?'
>>> antwoord
42
```

UNIVERSITEIT GENT

Alternatieve imports




```
>>> vraag = 'What is the air-speed velocity of an unladen swallow?'
>>> antwoord = 'What do you mean? An African or European swallow?'
>>> from module1 import *
>>> vraag
'What is the meaning of life, the Universe, and everything?'
>>> antwoord
42
```

"Bijna altijd een slecht idee."

UNIVERSITEIT GENT

Namespaces

- **namespace**: syntactische container die toelaat zelfde naam te gebruiken in modules, functies, methodes en klassen
- functies hebben ook eigen namespace

namespace.py

```
def f():
    n = 7
    print(f'n binnen f: {n}')

def g():
    n = 42
    print(f'n binnen g: {n}')

n = 11
print(f'n voor aanroep f: {n}')
f()
print(f'n na aanroep f: {n}')
g()
print(f'n na aanroep g: {n}')
```

```
$ python namespace.py
n voor aanroep f: 11
n binnen f: 7
n na aanroep f: 11
n binnen g: 42
n na aanroep g: 11
$
```

Import voert statements uit

- **`import`** is een statement
  - wordt uitgevoerd als Python het tegenkomt
  - gedraagt zich net als elk ander statement
- statements in module worden uitgevoerd bij opladen ervan
  - meestal enkel toekenningen en **`def`** statements
  - controlelussen, voorwaardelijke opdrachten en andere statements kunnen ook gebruikt worden in module

UNIVERSITEIT GENT

Import voert statements uit

```
>>> import drie_vragen
What is your name?
Sir Galahad of Camelot.
What is your quest?
I seek the Holy Grail.
What is your favorite color?
Blue.  No yell--
Auuuuuuuugh!
>>>
```

drie_vragen.py

```
antwoord1 = input('What is your name?\n')
antwoord2 = input('What is your quest?\n')
antwoord3 = input('What is your favorite color?\n')
if antwoord3.lower()[:6] == 'yellow':
    print('Right. Off you go.')
else:
    print('Auuuuuuuugh!')
```

UNIVERSITEIT GENT

Lunar phases

Maanfasen

$$d_n = d_r - p * \left\lfloor \frac{d_r}{p} \right\rfloor$$

aantal dagen sinds laatste nieuwe maan

aantal dagen verstreken sinds referentiedatum (6 januari 2000)

Test-driven development

Test-driven development (TDD) is een ontwikkelingsmethode voor het schrijven software waarbij eerst de testcode wordt geschreven en pas daarna de code die het probleem oplost.

Doctests

matrix1.py

```
def maakMatrix(rijen, kolommen):
    """
    Geeft een matrix terug met een gegeven aantal rijen
    en kolommen. Alle matrixelementen worden op 0 gezet.

    >>> maakMatrix(3, 5)
    [[0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0]]
    """

if __name__ == '__main__':
    import doctest
    doctest.testmod()
```

```
>>> from matrix1 import maakMatrix
>>> maakMatrix(3, 5)
>>> print(maakMatrix(3, 5))
None
>>>
```

UNIVERSITEIT GENT

Doctests

```
$ python matrix1.py
**********************************************************************
File "matrix1.py", line 6, in __main__.maakMatrix
Failed example:
    maakMatrix(3, 5)
Expected:
    [[0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0]]
Got nothing
**********************************************************************
1 items had failures:
   1 of   1 in __main__.maakMatrix
***Test Failed*** 1 failures.
$
```

UNIVERSITEIT GENT

Doctests

matrix2.py

```
def maakMatrix(rijen, kolommen):
    """
    Geeft een matrix terug met een gegeven aantal rijen
    en kolommen. Alle matrixelementen worden op 0 gezet.

    >>> maakMatrix(3, 5)
    [[0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0]]
    """
    return [[0,0,0,0,0], [0,0,0,0,0], [0,0,0,0,0]]

if __name__ == '__main__':
    import doctest
    doctest.testmod()
```

```
$ python matrix2.py
$
```

UNIVERSITEIT GENT

Doctests

matrix3.py

```
def maakMatrix(rijen, kolommen):
    """
    Geeft een matrix terug met een gegeven aantal rijen
    en kolommen. Alle matrixelementen worden op 0 gezet.

    >>> maakMatrix(3, 5)
    [[0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0]]
    >>> maakMatrix(4, 2)
    [[0, 0], [0, 0], [0, 0], [0, 0]]
    """
    return [[0,0,0,0,0], [0,0,0,0,0], [0,0,0,0,0]]

if __name__ == '__main__':
    import doctest
    doctest.testmod()
```

UNIVERSITEIT GENT

Doctests

```
$ python matrix3.py
**********************************************************************
File "matrix3.py", line 8, in __main__.maakMatrix
Failed example:
    maakMatrix(4, 2)
Expected:
    [[0, 0], [0, 0], [0, 0], [0, 0]]
Got:
    [[0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0]]
**********************************************************************
1 items had failures:
   1 of   2 in __main__.maakMatrix
***Test Failed*** 1 failures.
$
```

UNIVERSITEIT GENT

Doctests

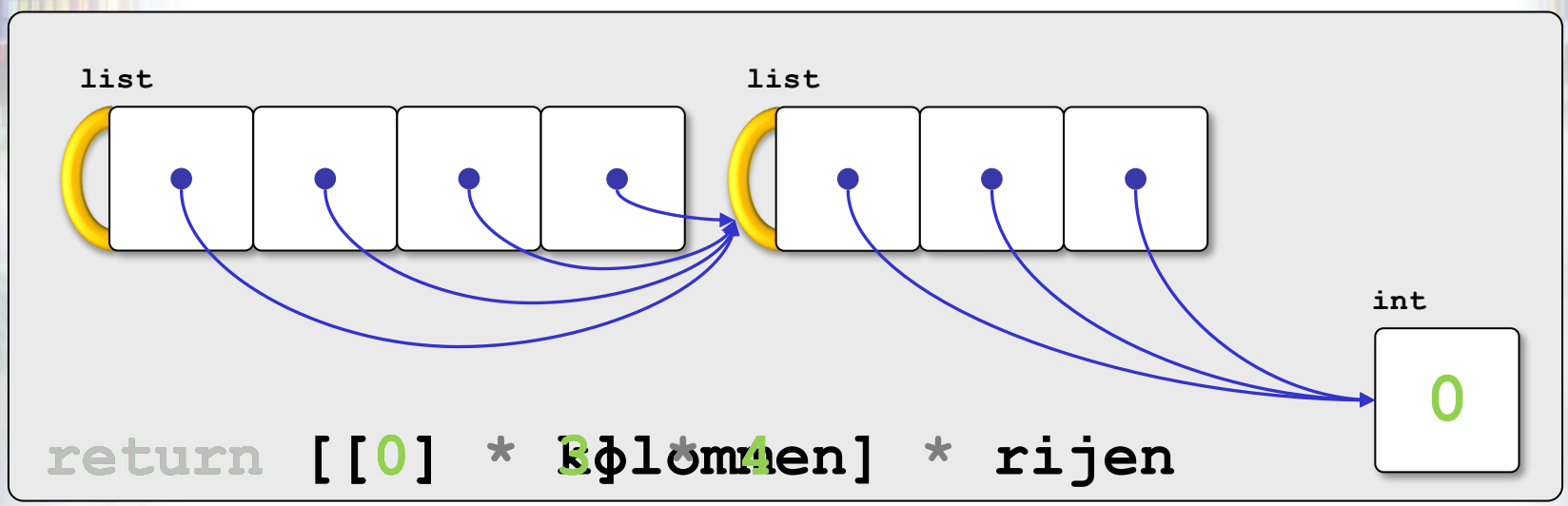
matrix4.py

```
def maakMatrix(rijen, kolommen):
    """
    Geeft een matrix terug met een gegeven aantal rijen
    en kolommen. Alle matrixelementen worden op 0 gezet.

    >>> maakMatrix(3, 5)
    [[0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0]]
    >>> maakMatrix(4, 2)
    [[0, 0], [0, 0], [0, 0], [0, 0]]
    """
    return [[0] * kolommen] * rijen

if __name__ == '__main__':
    import doctest
    doctest.testmod()
```

```
$ python matrix4.py
$
```

UNIVERSITEIT GENT

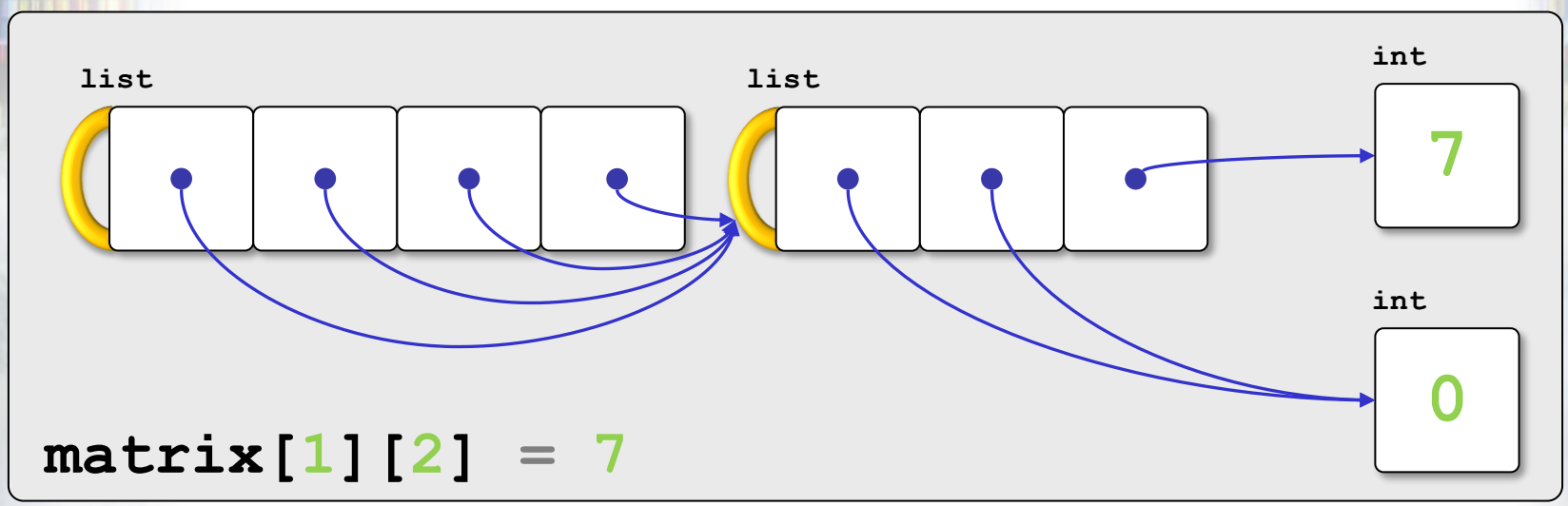
Doctests




```
>>> from matrix4 import *
>>> matrix = maakMatrix(4, 3)
>>> matrix
[[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0]]
>>> matrix[1][2] = 7
>>> matrix
[[0, 0, 7], [0, 0, 7], [0, 0, 7], [0, 0, 7]]
```

UNIVERSITEIT GENT

Doctests




```
>>> from matrix4 import *
>>> matrix = maakMatrix(4, 3)
>>> matrix
[[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0]]
>>> matrix[1][2] = 7
>>> matrix
[[0, 0, 7], [0, 0, 7], [0, 0, 7], [0, 0, 7]]
```

UNIVERSITEIT GENT

Doctests

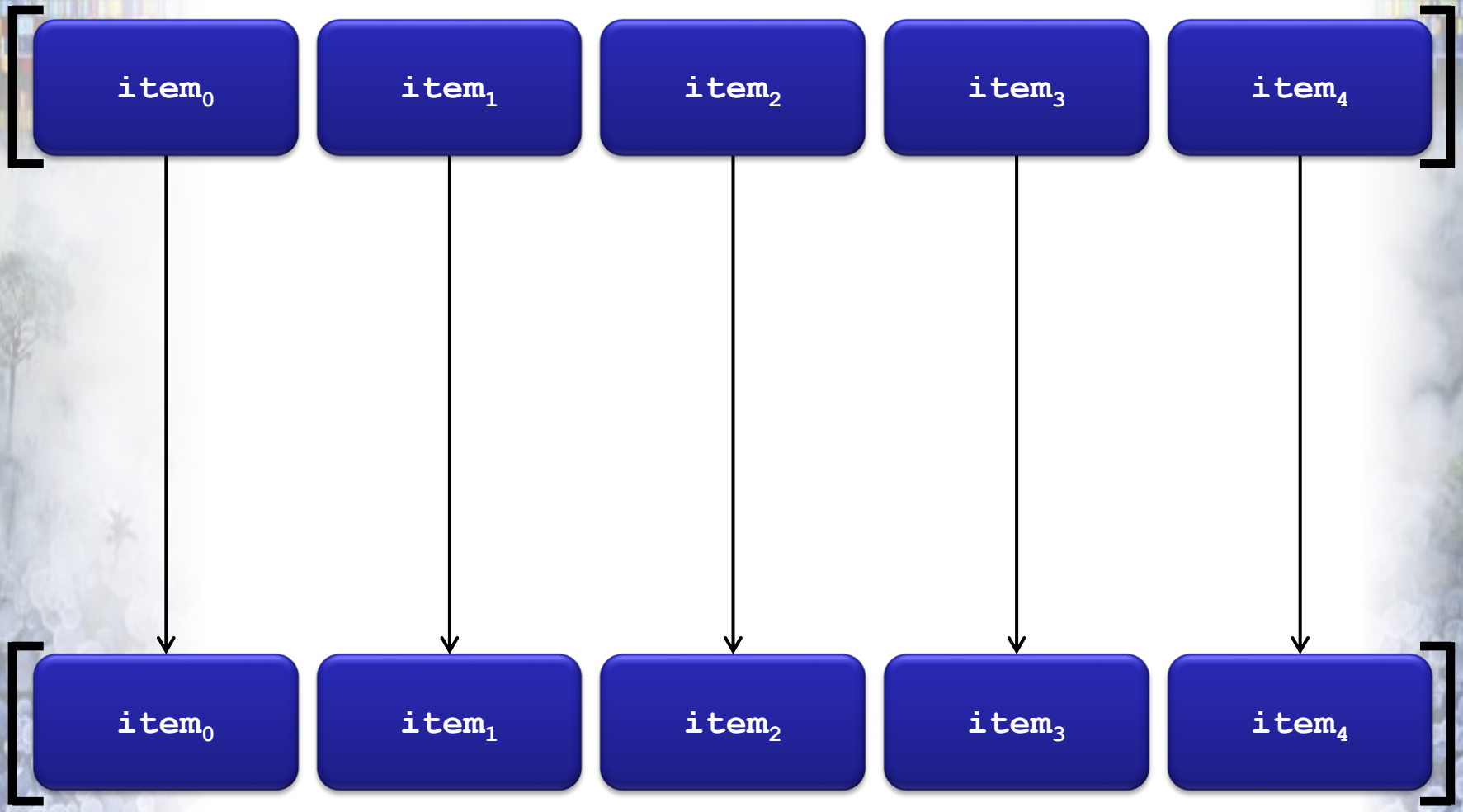
matrix5.py

```
def maakMatrix(rijen, kolommen):
    """
    Geeft een matrix terug met een gegeven aantal rijen
    en kolommen. Alle matrixelementen worden op 0 gezet.

    >>> maakMatrix(3, 5)
    [[0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0]]
    >>> maakMatrix(4, 2)
    [[0, 0], [0, 0], [0, 0], [0, 0]]
    >>> matrix = maakMatrix(4, 2)
    >>> matrix[1][1] = 7
    >>> matrix
    [[0, 0], [0, 7], [0, 0], [0, 0]]
    """
    matrix = []
    for rij in range(rijen):
        matrix += [[0] * kolommen]
    return matrix

if __name__ == '__main__':
    import doctest
    doctest.testmod()
```

UNIVERSITEIT GENT

List comprehensions

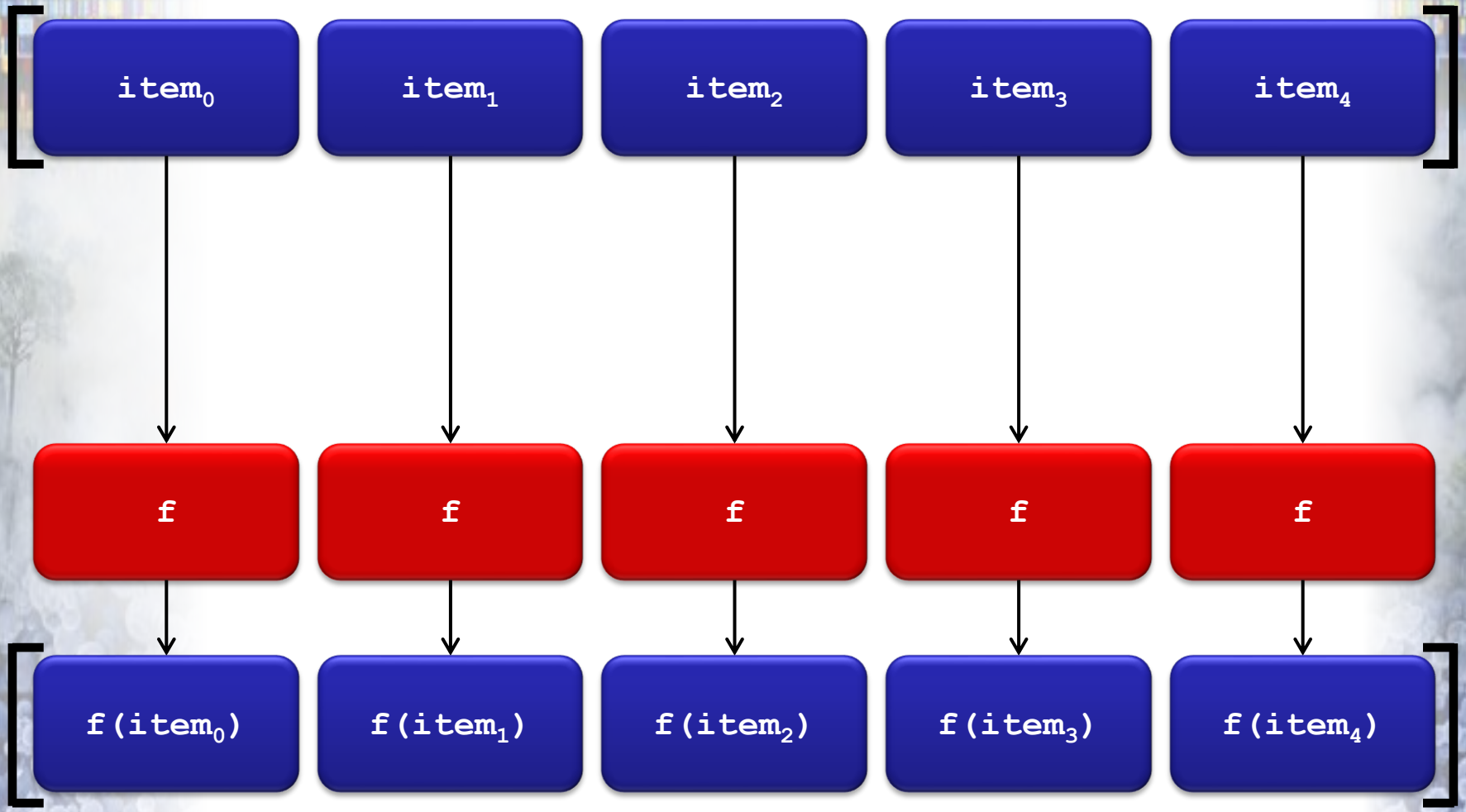
List comprehensions



UNIVERSITEIT GENT

```
nieuwelijst = [item for item in lijst]
```

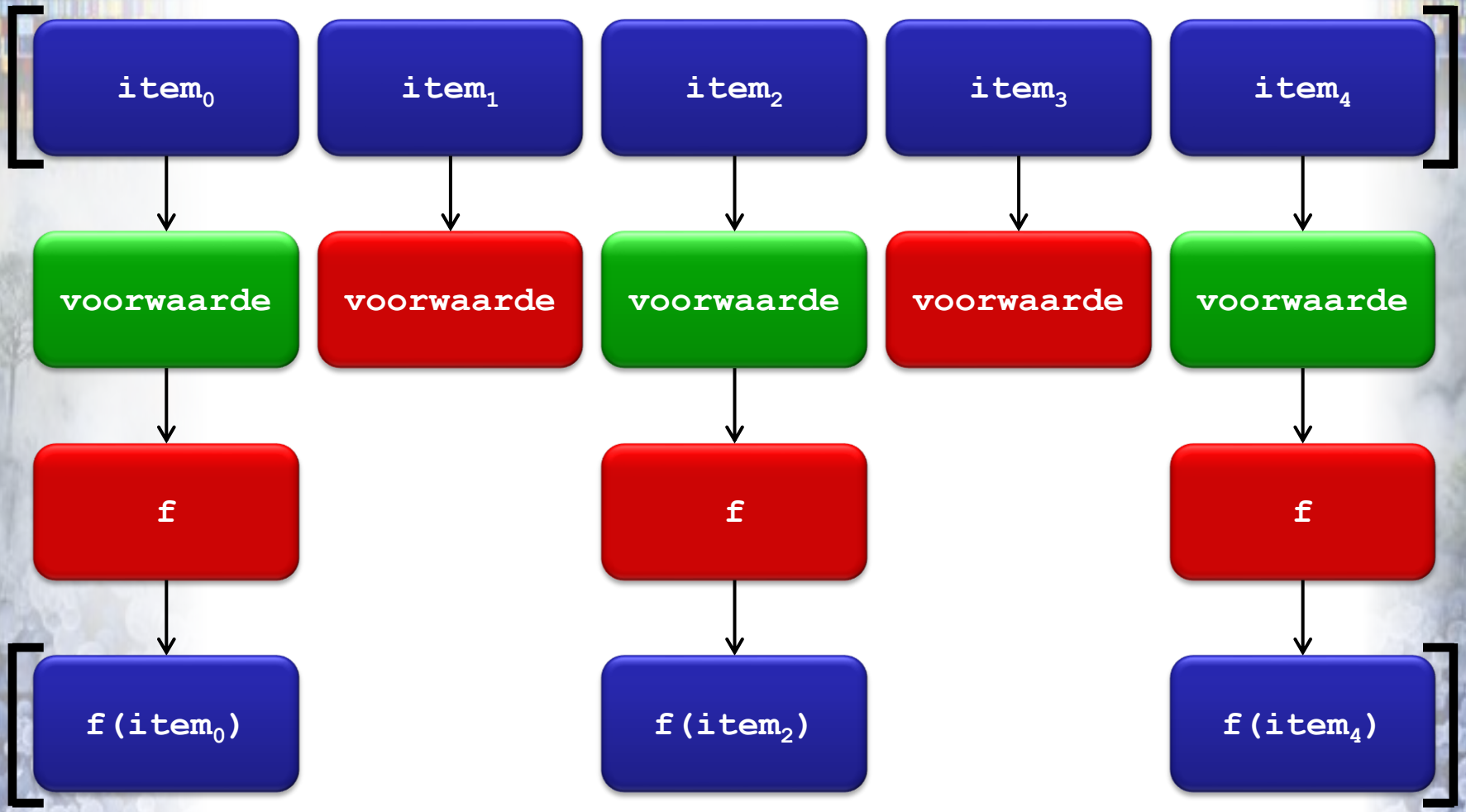
List comprehensions



UNIVERSITEIT GENT

```
nieuwelijst = [f(item) for item in lijst]
```

List comprehensions



UNIVERSITEIT GENT

```
nieuwelijst = [f(item) for item in lijst if voorwaarde]
```

List comprehensions

- zeer krachtig taalelement
  - nieuwe lijst aanmaken door expressie toe te passen op selectie van elementen uit samengesteld gegevenstype
  - heel vaak gebruikt door Python programmeurs
    - veel voorbeelden te vinden in bestaande code

- vrij ongebruikelijke syntaxis
  - lijkt op syntaxis van **`for`** controlelus, **`in`** operator of **`if`** statement
  - deze drie sleutelwoorden (**`for`**, **`in`** en **`if`**) worden ook (soms zelfs herhaaldelijk) gebruikt bij het construeren van list comprehensions

UNIVERSITEIT GENT

List comprehensions

- basisvorm

```
[ expressie for variabele in iterable ]
```

  - **`expressie`** is bewerking die wordt uitgevoerd op **`variabele`**
  - voor elk element van **`iterable`** zal de list comprehension
    1. **`variabele`** laten verwijzen naar dit element
    2. nieuwe waarde berekenen als evaluatie van **`expressie`**
       - expressies die resulteren in tuple moeten tussen ronde haakjes staan
  - nieuwe waarden worden verzameld in lijst die als resultaat van list comprehension wordt teruggegeven

```
>>> li = [3, 6, 2, 7]
>>> [2 * elem for elem in li]
[6, 12, 4, 14]
```

UNIVERSITEIT GENT

List comprehensions

- basisvorm

```
[ expressie for variabele in iterable ]
```

  - **`iterable`** kan elementen van verschillende types bevatten
    - **`expressie`** moet geldig zijn voor elk element van **`iterable`**
  - elementen van **`iterable`** kunnen zelf containers zijn
    - **`variabele`** is container van variabelen met gegevenstype en "vorm" die corresponderen met **`iterable`**-elementen

```
>>> li = [3, 'spam', [4, 5]]
>>> [3 * elem for elem in li]
[9, 'spamspamspam', [4, 5, 4, 5, 4, 5]]
>>> li = [('a', 1), ('b', 2), ('c', 7)]
>>> [3 * n for x, n in li]
[3, 6, 21]
```

UNIVERSITEIT GENT

List comprehensions

- basisvorm

```
[ expressie for variabele in iterable ]
```

  - **`expressie`** kan ook zelfgedefinieerde functie bevatten
    - **`expressie`** moet geldig zijn voor elk element van **`iterable`**

```
>>> def aftrekken(a, b):
...     return a - b
...
>>> li = [(6, 3), (1, 7), (5, 5)]
>>> [aftrekken(y, x) for x, y in li]
[-3, 6, 0]
```

UNIVERSITEIT GENT

Bijkomende filtering

- uitgebreide vorm

```
[ expressie for variabele1 in iterable1
  [ if voorwaarde | for variabele2 in iterable2 ] … ]
```

  - **voorwaarde** bepaalt of **expressie** wordt toegepast op gegeven element uit **iterable**
    - **voorwaarde** wordt gecontroleerd voor elk element van **iterable** in voorgaande **for** component van comprehension
    - elementen waarvoor voorwaarde niet geldt worden weggelaten uit lijst vóór evaluatie van **expressie** uit list comprehension

```
>>> li = [3, 6, 2, 7, 1, 9]
>>> [2 * elem for elem in li if elem > 4]
[12, 14, 18]
```

UNIVERSITEIT GENT

Bijkomende filtering

```
>>> '1 2 3 4'.split()
['1', '2', '3', '4']
>>> [int(x) for x in '1 2 3 4'.split()]   # conversie
[1, 2, 3, 4]
>>> [x * x for x in range(10)]            # macht
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
>>> [x for x in range(10) if not x % 2]   # even getallen
[0, 2, 4, 6, 8]
>>> [(b, a) for a, b in [(1, 2), (3, 4)]] # omwisselen
[(2, 1), (4, 3)]
>>> s = "monty python's flying circus"
>>> [ord(c) for c in s]
[109, 111, 110, 116, 121, 32, 112, 121, 116, 104, … ]
>>> [i for i, c in enumerate(s) if c == ' ']
[5, 14, 21]
>>> [p for p in range(2, 100)
...      if not [x for x in range(2, p) if not p % x]]
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, …, 97]
```

UNIVERSITEIT GENT

List comprehensions nesten

- list comprehension neemt iterable als invoer en genereert lijst als uitvoer
  - lijst is zelf ook een iterable
  - mogelijk om list comprehensions te nesten

- voorbeeld
  - binnenste comprehension produceert lijst `[4, 3, 5, 2]`
  - buitenste comprehension produceert lijst `[8, 6, 10, 4]`

```
>>> li = [3, 2, 4, 1]
>>> [2 * elem for elem in [item + 1 for item in li]]
[8, 6, 10, 4]
```

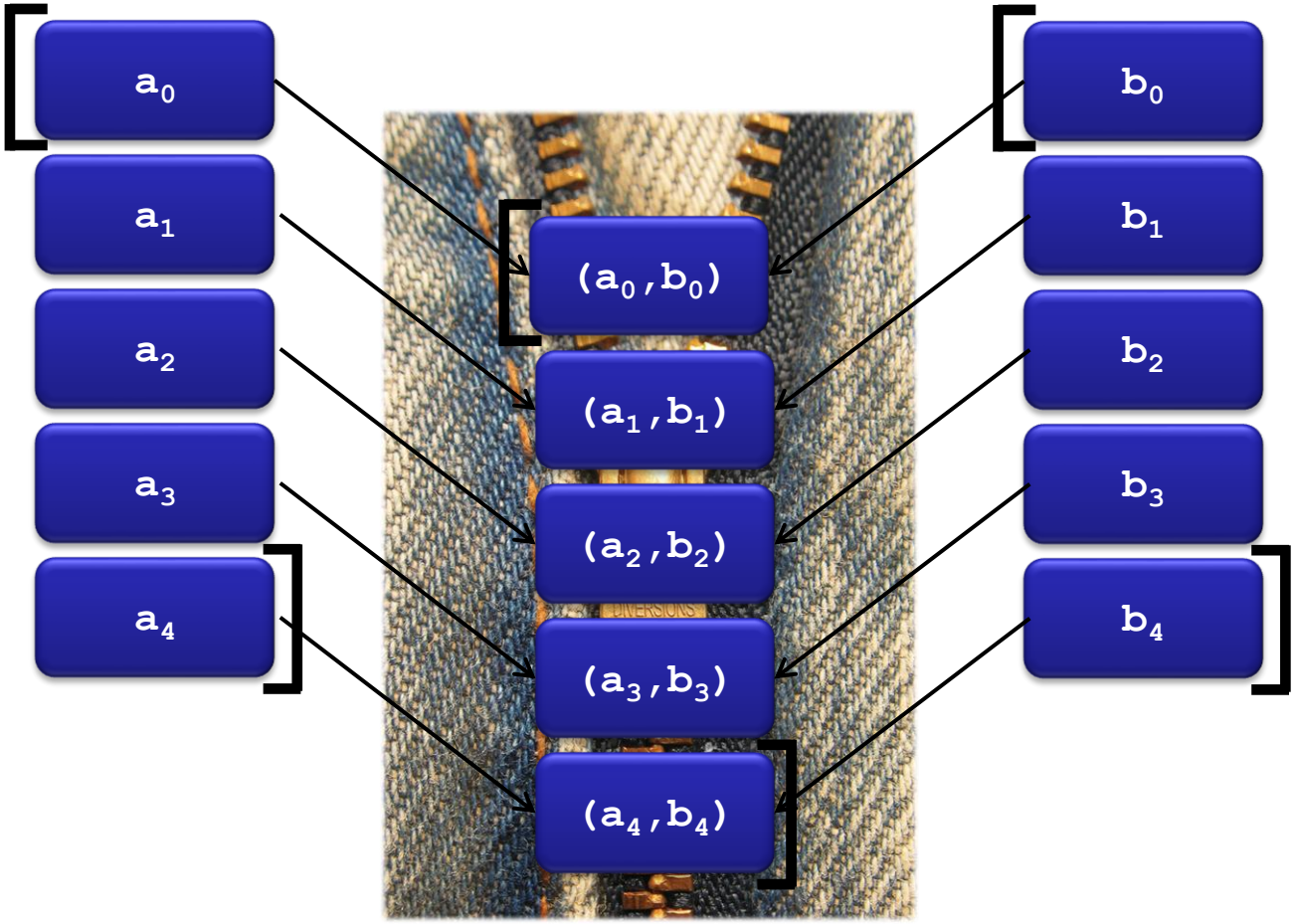
UNIVERSITEIT GENT

List comprehensions nesten

```
>>> vec = [2, 4, 6]
>>> [[x, x**2] for x in vec]
[[2, 4], [4, 16], [6, 36]]
>>> [x, x**2 for x in vec]      # tuple vereist ronde haakjes
  File "<stdin>", line 1
    [x, x**2 for x in vec]
               ^
SyntaxError: invalid syntax
>>> [(x, x**2) for x in vec]
[(2, 4), (4, 16), (6, 36)]
>>> vec1 = [2, 4, 6]
>>> vec2 = [4, 3, -9]
>>> [x * y for x in vec1 for y in vec2]
[8, 6, -18, 16, 12, -36, 24, 18, -54]
>>> [x + y for x in vec1 for y in vec2]
[6, 5, -7, 8, 7, -5, 10, 9, -3]
>>> [vec1[i] * vec2[i] for i in range(len(vec1))]
[8, 12, -54]
```

UNIVERSITEIT GENT

Zip



UNIVERSITEIT GENT

Zip

- combineert twee of meer *iterable objects* door corresponderende elementen te bundelen tot tuples van een nieuwe iterable

```
>>> voornamen = ['John', 'Terry', 'Eric']
>>> familienamen = ['Cleese', 'Gilliam', 'Idle']
>>> namen = list(zip(voornamen, familienamen))
>>> namen
[('John', 'Cleese'), ('Terry', 'Gilliam'), ('Eric', 'Idle')]
>>> list(zip(*namen))      # ook voor meer dan één lijst
[('John', 'Terry', 'Eric'), ('Cleese', 'Gilliam', 'Idle')]
                           # * bij argument resulteert in
                           # zip(namen[0],namen[1],namen[2])
```

UNIVERSITEIT GENT

Huiswerk (volgende les)

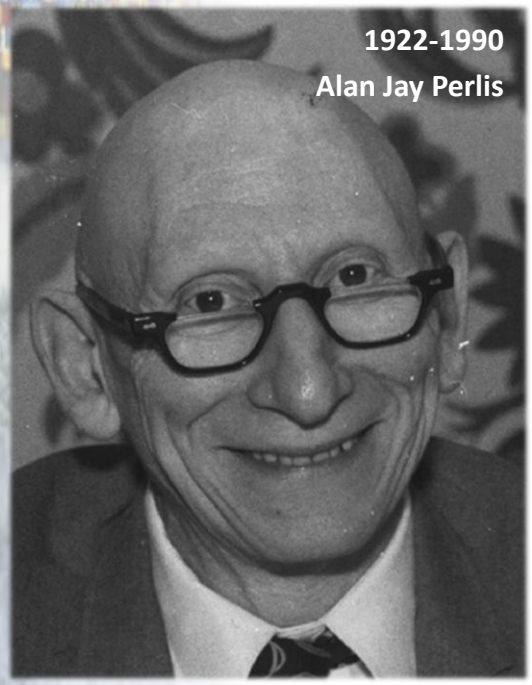
- handboek
  - lees hoofdstuk 9 (sets en dictionaries)
  - lees hoofdstuk 10 (more program development)
- voorbeeldopgaven volgende les (reeks 8)
  - Polymeren simuleren
  - Bloedgroepen
  - Cryptogrammen

UNIVI
GENT

Vragen of opmerkingen?

UNIVERSITEIT GENT

The sky is the limit…



"If a listener nods his head when you're explaining your program, wake him up."

— Alan J. Perlis

UNIVERSITEIT GENT