Er liggen vier kaarten op een tafel. Elke kaart heeft een getal op één zijde en een kleur op de ander zijde.

Welke kaart(en) moet je zeker omdraaien om de uitspraak te bevestigen dat als op de ene zijde van een kaart een even getal staat, de andere zijde een rode kleur moet hebben ?

Universiteit Gent
Faculteit Wetenschappen

Programmeren in Python

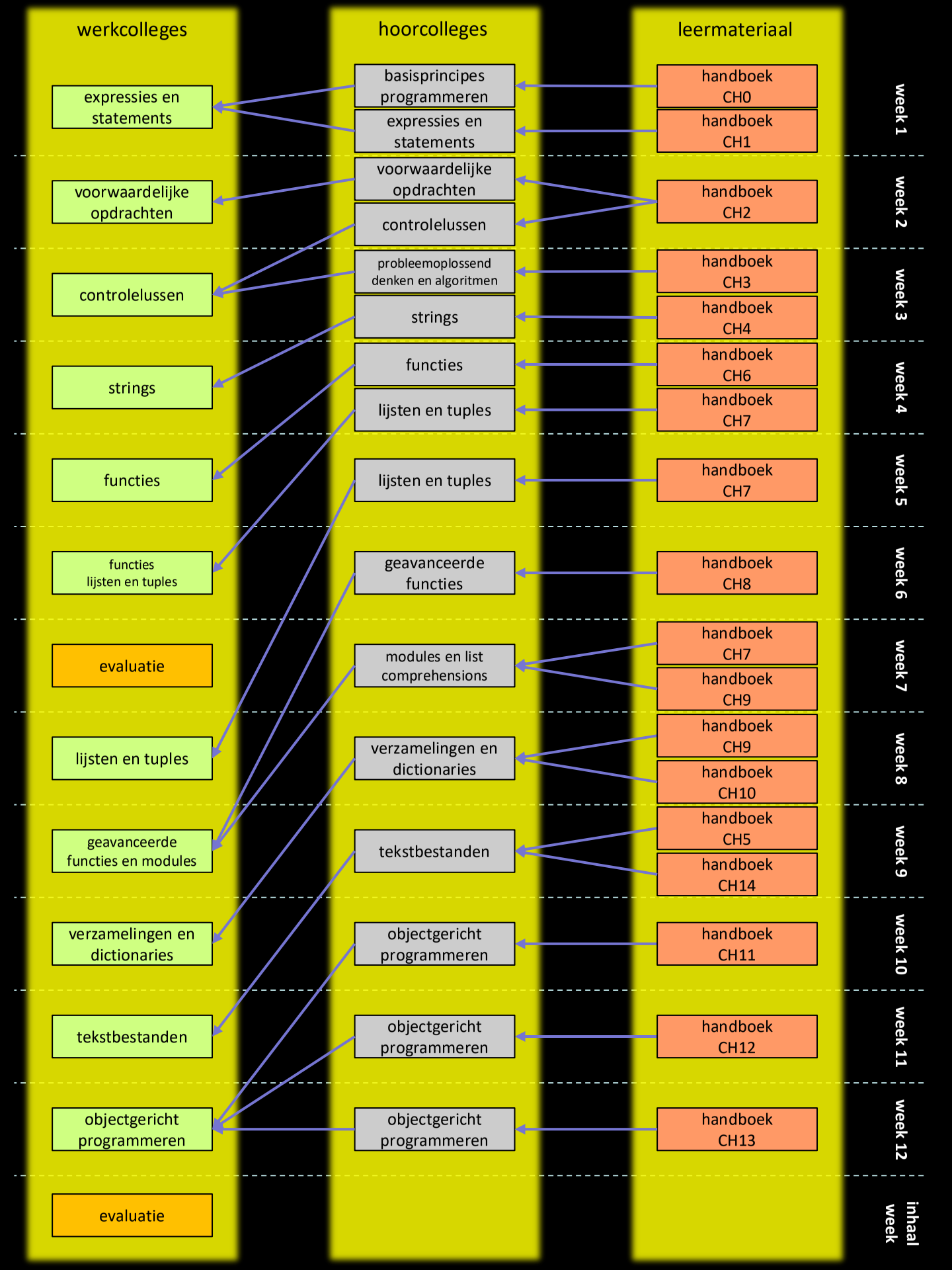
go with the flow

prof. dr. Peter Dawyndt

peter.dawyndt@ugent.be

@dawyndt

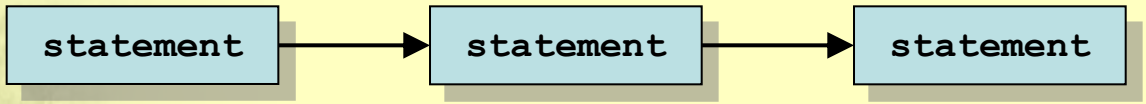
UNIVERSITEIT GENT

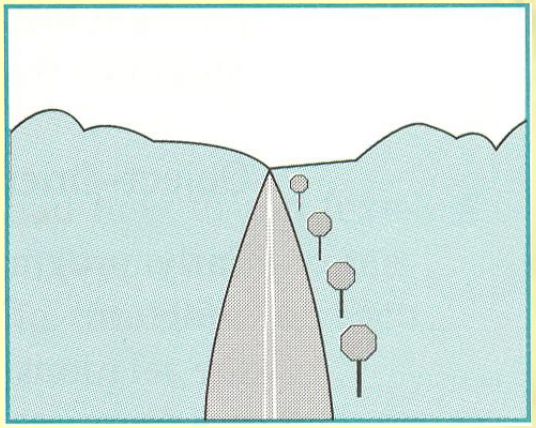

werkcolleges
hoorcolleges
leermateriaal
expressies en statements
voorwaardelijke opdrachten
controlelussen
strings
functies
functies lijsten en tuples
evaluatie
lijsten en tuples
geavanceerde functies en modules
verzamelingen en dictionaries
tekstbestanden
objectgericht programmeren
evaluatie
basisprincipes 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

Salade van peer en appel met

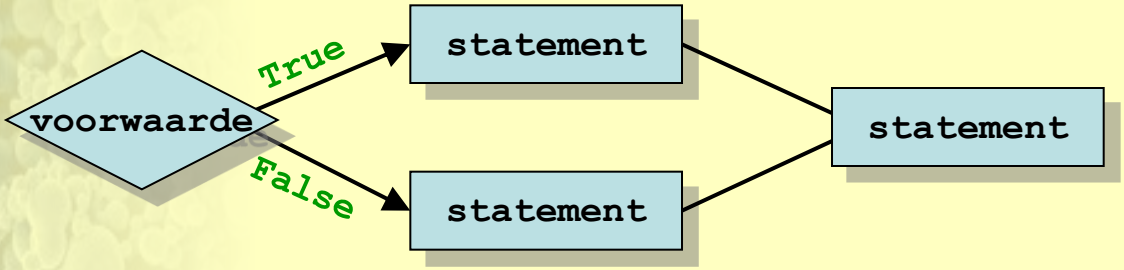
Controlestructuren

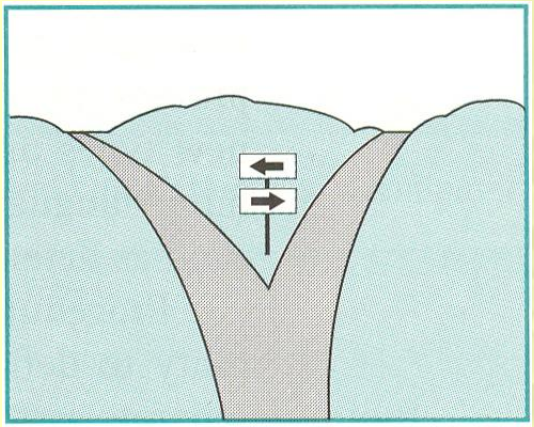
sequentie





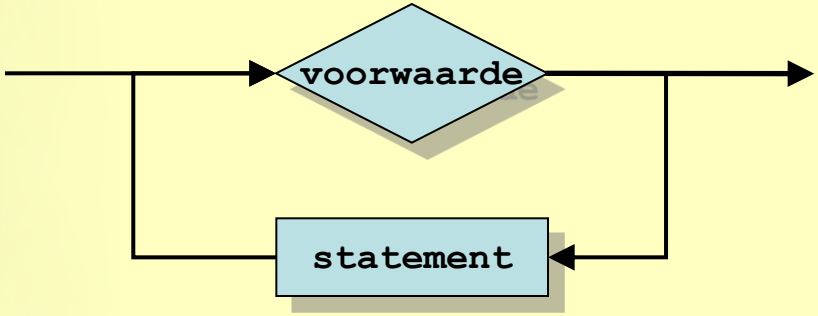
voorwaardelijke opdracht (selectie, sprong)

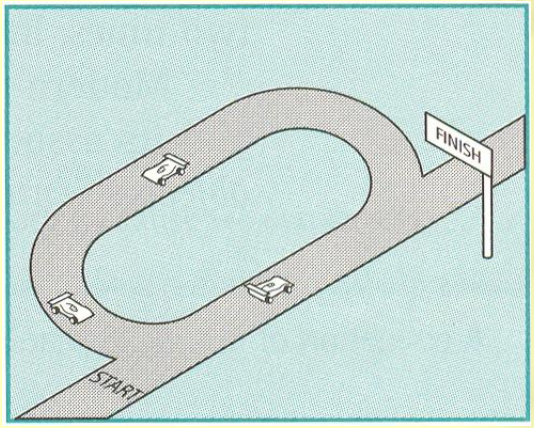




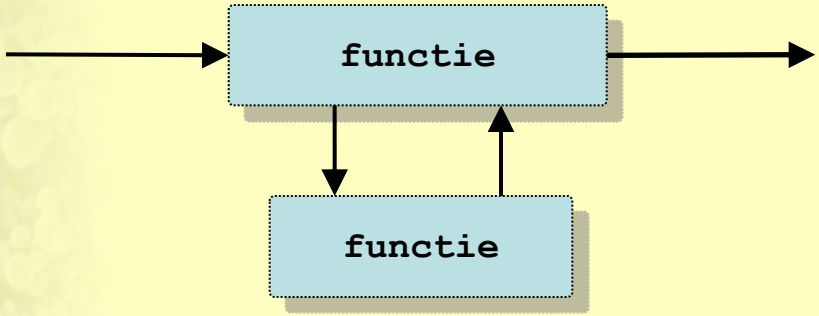
Controlestructuren

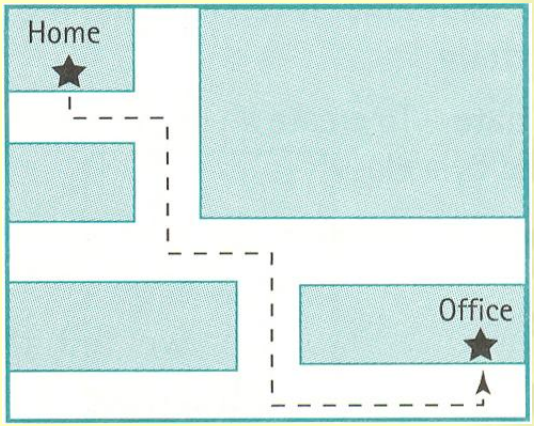
repetitieve opdracht (lus, herhaling, iteratie)





functie (methode)





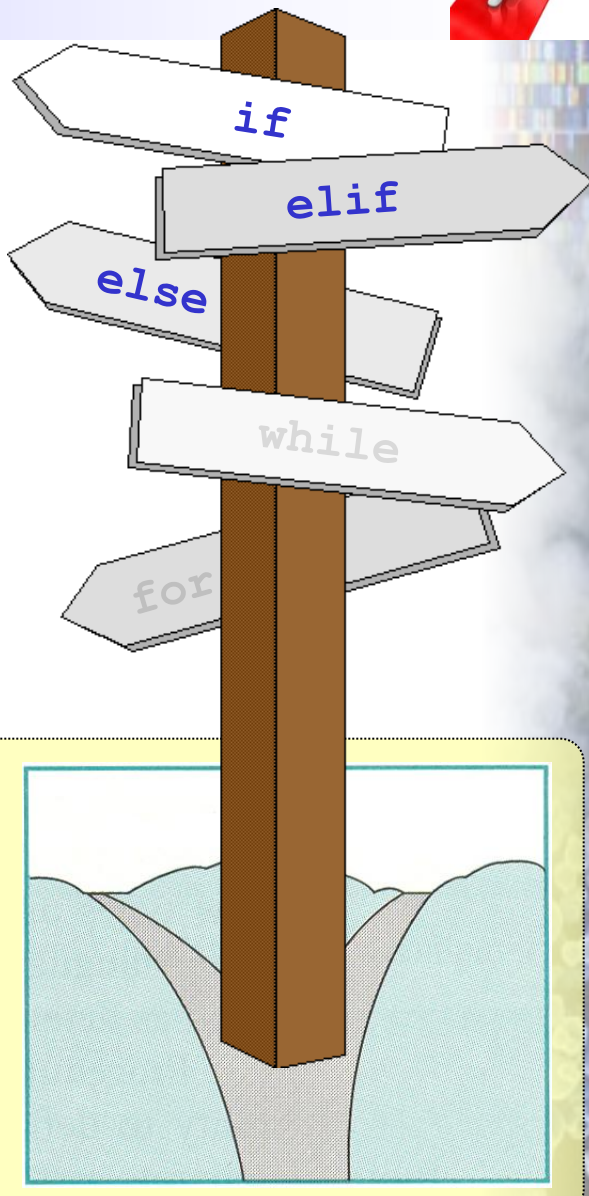
Controlestructuren

- voorwaardelijke opdrachten
- repetitieve opdrachten
- functies

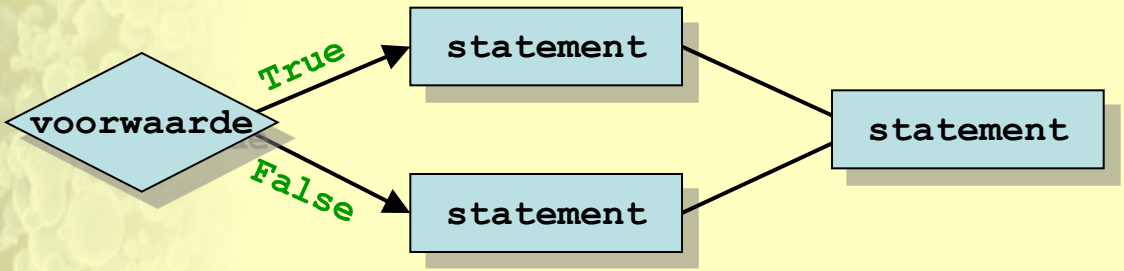
UNIVERSITEIT GENT

Controlestructuren

- voorwaardelijke opdrachten
- repetitieve opdrachten
- functies



voorwaardelijke opdracht (selectie, sprong)



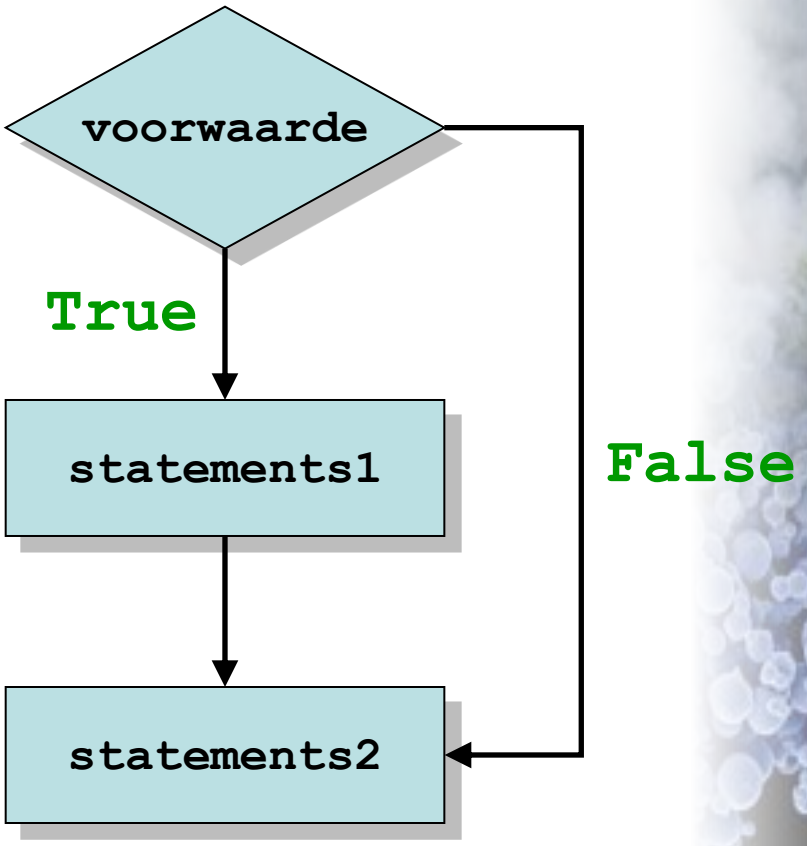
Voorwaardelijk uitvoeren

syntaxis

```
if booleaanse expressie:
    statements
```

voorbeeld

```
if x > 0:
    print('x is positief')
```



UNIVERSITEIT GENT

Pass statement

- aantal statements in body van **`if`** statement is onbegrensd
  - er moet er minstens één staan
  - body "zonder statements" is soms handig
    - bv. als pas later broncode zal aangevuld worden
  - **`pass`** statement
    - statement dat verder niets doet

```
if booleaanse expressie:
    statements
```

UNIVERSITEIT GENT

Pass statement

- aantal statements in body van **`if`** statement is onbegrensd
  - er moet er minstens één staan
  - body "zonder statements" is soms handig
    - bv. als pas later broncode zal aangevuld worden
  - **`pass`** statement
    - statement dat verder niets doet

UNIVERSITEIT GENT

```
if True:        # dit is altijd waar
    pass        # altijd uitgevoerd, maar doet niets
```

Modulorekenen

- modulo operator (%)
  - werkt zowel op integers als floats
  - rest na gehele deling van eerste door tweede operand

```
>>> quotient = 7 // 3
>>> print(quotient)
2
>>> rest = 7 % 3
>>> print(rest)
1
```

UNIVERSITEIT GENT

Modulorekenen

- modulo operator (`%`)
  - werkt zowel op integers als floats
  - rest na gehele deling van eerste door tweede operand
- handige operator om testen uit te voeren
  - nagaan of **`x`** deelbaar is door **`y`**
  - bepaal cijfers aan rechterkant van integer

```
>>> x = 18; y = 3; z = 1234
>>> x % y == 0
True
>>> z % 10              # eenheden
4
>>> z % 100             # tientallen
34
```

UNIVERSITEIT GENT

Booleaanse waarden

- gegevenstype **`bool`**
  - stelt waarden *waar* (**`True`**) en *onwaar* (**`False`**) voor
    - *case sensitive*
  - genoemd naar Britse wiskundige George Boole
    - grondlegger van de Booleaanse algebra

```
>>> type(True)
<class 'bool'>
>>> type(False)
<class 'bool'>
>>> type(true)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'true' is not defined
```

UNIVERSITEIT GENT

Booleaanse expressies

- Booleaanse expressie
  - expressie die Booleaanse waarde oplevert
  - operator **==** vergelijkt twee waarden
    - levert Booleaanse waarde op
    - voorbeeld van *vergelijkingsoperator*

```
>>> 5 == 5
True
>>> 5 == 6
False
>>> x = (5 == 5)
>>> print(x)
True
```

UNIVERSITEIT GENT

Vergelijkingsoperatoren

operator	betekenis
`x == y`	is gelijk aan
`x != y`	is verschillend van
`x < y`	is kleiner dan
`x > y`	is groter dan
`x <= y`	is kleiner dan of gelijk aan
`x >= y`	is groter dan of gelijk aan

UNIVERSITEIT GENT

Logische operatoren

and	True	False
True	True	False
False	False	False

conjunctie

or	True	False
True	True	True
False	True	False

disjunctie

	not
True	False
False	True

negatie

UNIVERSITEIT GENT


AND
Truth table
A
B
Output
0
0
0
0
1
0
1
1
1

Logische operatoren

operator	benaming	betekenis
`and`	conjunctie	levert `True` op als beide voorwaarden waar zijn, en `False` in alle andere gevallen
`or`	disjunctie	levert `False` op als beide voorwaarden onwaar zijn, en `True` in alle andere gevallen
`not`	negatie	Zet `False` om in `True`, en `True` in `False`

UNIVERSITEIT GENT

Logische operatoren

operator	benaming	betekenis
`and`	conjunctie	levert `True` op als beide voorwaarden waar zijn, en `False` in alle andere gevallen
`or`	disjunctie	levert `False` op als beide voorwaarden onwaar zijn, en `True` in alle andere gevallen
`not`	negatie	Zet `False` om in `True`, en `True` in `False`

```
>>> leeftijd = 12
>>> if leeftijd >= 6 and leeftijd <= 18:
...     print('Je bent schoolplichtig!!')
...
Je bent schoolplichtig!!
```

UNIVERSITEIT GENT

Logische operatoren

operator	benaming	betekenis
`and`	conjunctie	levert `True` op als beide voorwaarden waar zijn, en `False` in alle andere gevallen
`or`	disjunctie	levert `False` op als beide voorwaarden onwaar zijn, en `True` in alle andere gevallen
`not`	negatie	Zet `False` om in `True`, en `True` in `False`

```
>>> leeftijd = 12
>>> if 6 <= leeftijd <= 18:
...     print('Je bent schoolplichtig!!')
...
Je bent schoolplichtig!!
```

UNIVERSITEIT GENT

Logische operatoren

operator	benaming	betekenis
`and`	conjunctie	levert `True` op als beide voorwaarden waar zijn, en `False` in alle andere gevallen
`or`	disjunctie	levert `False` op als beide voorwaarden onwaar zijn, en `True` in alle andere gevallen
`not`	negatie	Zet `False` om in `True`, en `True` in `False`

```
>>> vandaag = 'zaterdag'
>>> if vandaag == 'zaterdag' or vandaag == 'zondag':
...     print('Het is weekend!!')
...
Het is weekend!!
```

UNIVERSITEIT GENT

Logische operatoren

operator	benaming	betekenis
`and`	conjunctie	levert `True` op als beide voorwaarden waar zijn, en `False` in alle andere gevallen
`or`	disjunctie	levert `False` op als beide voorwaarden onwaar zijn, en `True` in alle andere gevallen
`not`	negatie	Zet `False` om in `True`, en `True` in `False`

```
>>> vandaag = 'zaterdag'
>>> if vandaag in {'zaterdag', 'zondag'}:
...     print('Het is weekend!!')
...
Het is weekend!!
```

UNIVERSITEIT GENT

Logische operatoren

operator	benaming	betekenis
`and`	conjunctie	levert `True` op als beide voorwaarden waar zijn, en `False` in alle andere gevallen
`or`	disjunctie	levert `False` op als beide voorwaarden onwaar zijn, en `True` in alle andere gevallen
`not`	negatie	Zet `False` om in `True`, en `True` in `False`

```
>>> vandaag = 'dinsdag'
>>> if not(vandaag == 'zaterdag' or vandaag == 'zondag'):
...     print('Vandaag is er les!!')
...
Vandaag is er les!!
```

UNIVERSITEIT GENT

Logische operatoren

operator	benaming	betekenis
`and`	conjunctie	levert `True` op als beide voorwaarden waar zijn, en `False` in alle andere gevallen
`or`	disjunctie	levert `False` op als beide voorwaarden onwaar zijn, en `True` in alle andere gevallen
`not`	negatie	Zet `False` om in `True`, en `True` in `False`

```
>>> vandaag = 'dinsdag'
>>> if vandaag != 'zaterdag' and vandaag != 'zondag':
...     print('Vandaag is er les!!')
...
Vandaag is er les!!
```

UNIVERSITEIT GENT

Logische operatoren

operator	benaming	betekenis
`and`	conjunctie	levert `True` op als beide voorwaarden waar zijn, en `False` in alle andere gevallen
`or`	disjunctie	levert `False` op als beide voorwaarden onwaar zijn, en `True` in alle andere gevallen
`not`	negatie	Zet `False` om in `True`, en `True` in `False`

```
>>> vandaag = 'dinsdag'
>>> if vandaag not in {'zaterdag', 'zondag'}:
...     print('Vandaag is er les!!')
...
Vandaag is er les!!
```

UNIVERSITEIT GENT

Logische operatoren

operator	benaming	betekenis
`and`	conjunctie	levert `True` op als beide voorwaarden waar zijn, en `False` in alle andere gevallen
`or`	disjunctie	levert `False` op als beide voorwaarden onwaar zijn, en `True` in alle andere gevallen
`not`	negatie	Zet `False` om in `True`, en `True` in `False`

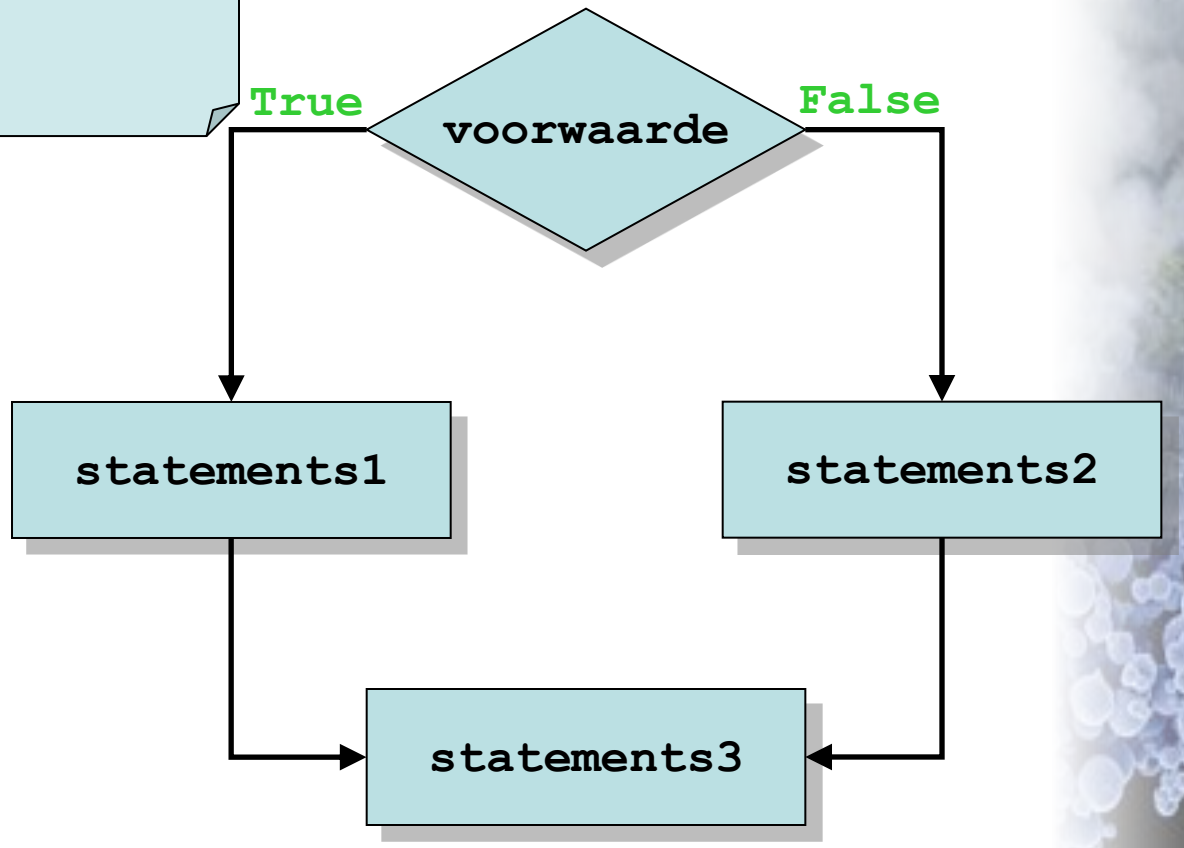
```
>>> vandaag = 'dinsdag'
>>> if not vandaag in {'zaterdag', 'zondag'}:
...     print('Vandaag is er les!!')
...
Vandaag is er les!!
```

UNIVERSITEIT GENT

Alternatieve uitvoering

syntaxis

```
if booleaanse expressie:
    statements1
else:
    statements2
```



UNIVERSITEIT GENT

Alternatieve uitvoering

syntaxis

```
if booleaanse expressie:
    statements1
else:
    statements2
```

```
>>> x = 4
>>> if x % 2 != 0:
...     print(f'{x} is oneven')
... else:
...     print(f'{x} is even')
...
4 is even
```

UNIVERSITEIT GENT

Alternatieve uitvoering

syntaxis

```
if booleaanse expressie:
    statements1
else:
    statements2
```

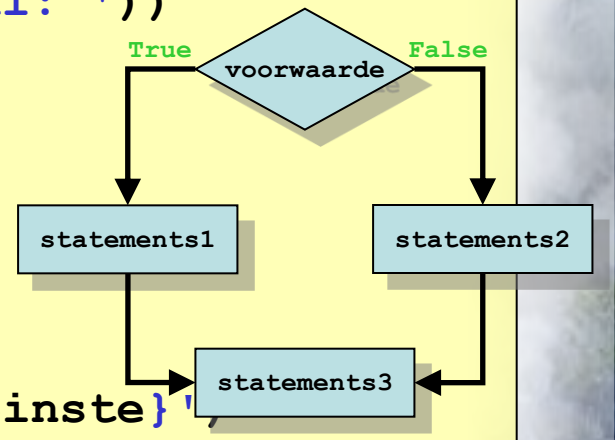
```
>>> x = 4
>>> if x % 2:
...     print(f'{x} is oneven')
... else:
...     print(f'{x} is even')
...
4 is even
```

UNIVERSITEIT GENT

Alternatieve uitvoering

kleinste.py

```
# bepaal kleinste van twee gegeven getallen
x = float(input('Geef een getal: '))
y = float(input('Geef nog een getal: '))

if x < y:
    kleinste = x
else:
    kleinste = y

print(f'Het kleinste getal is {kleinste}')
```

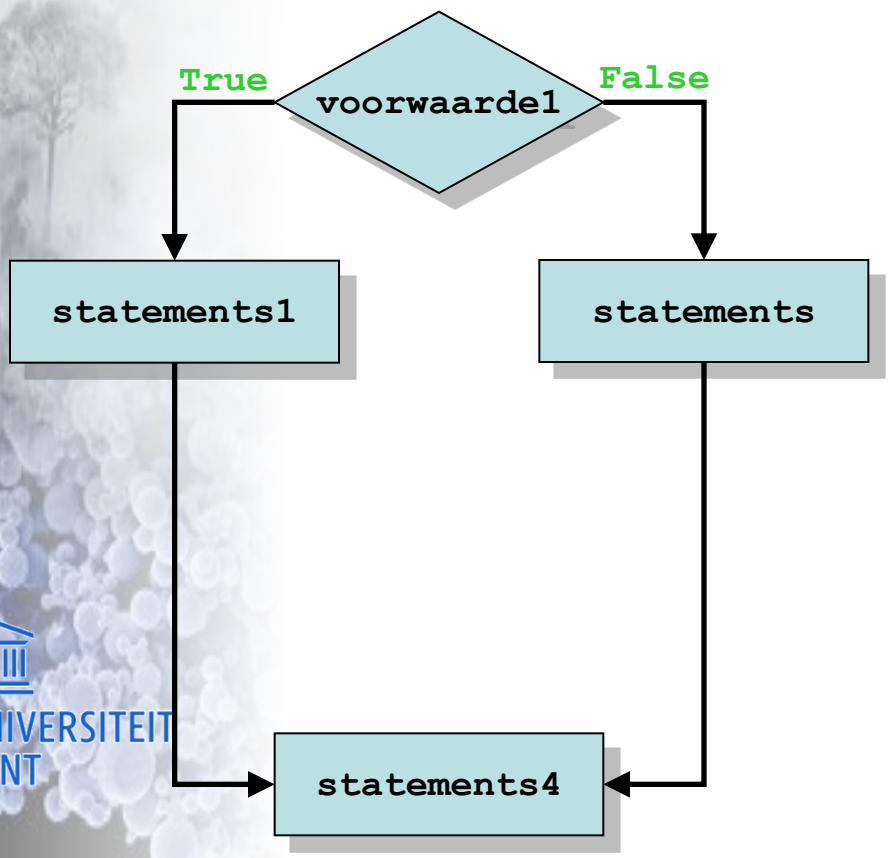


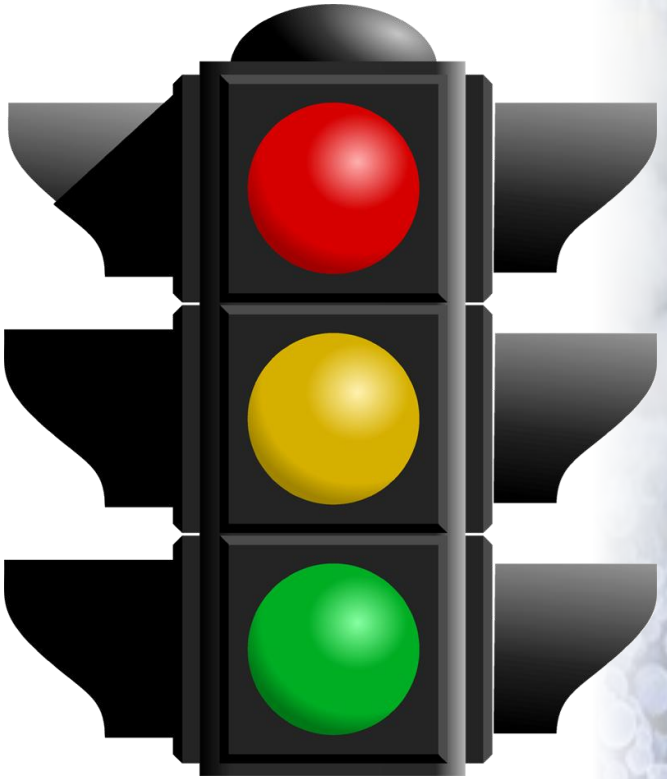

```
$ python3 kleinste.py
Geef een getal: 3.12
Geef nog een getal: 7.83
Het kleinste getal is 3.12
```

Wat zal er gebeuren als x en y gelijk zijn aan elkaar ?

UNIVERSITEIT GENT

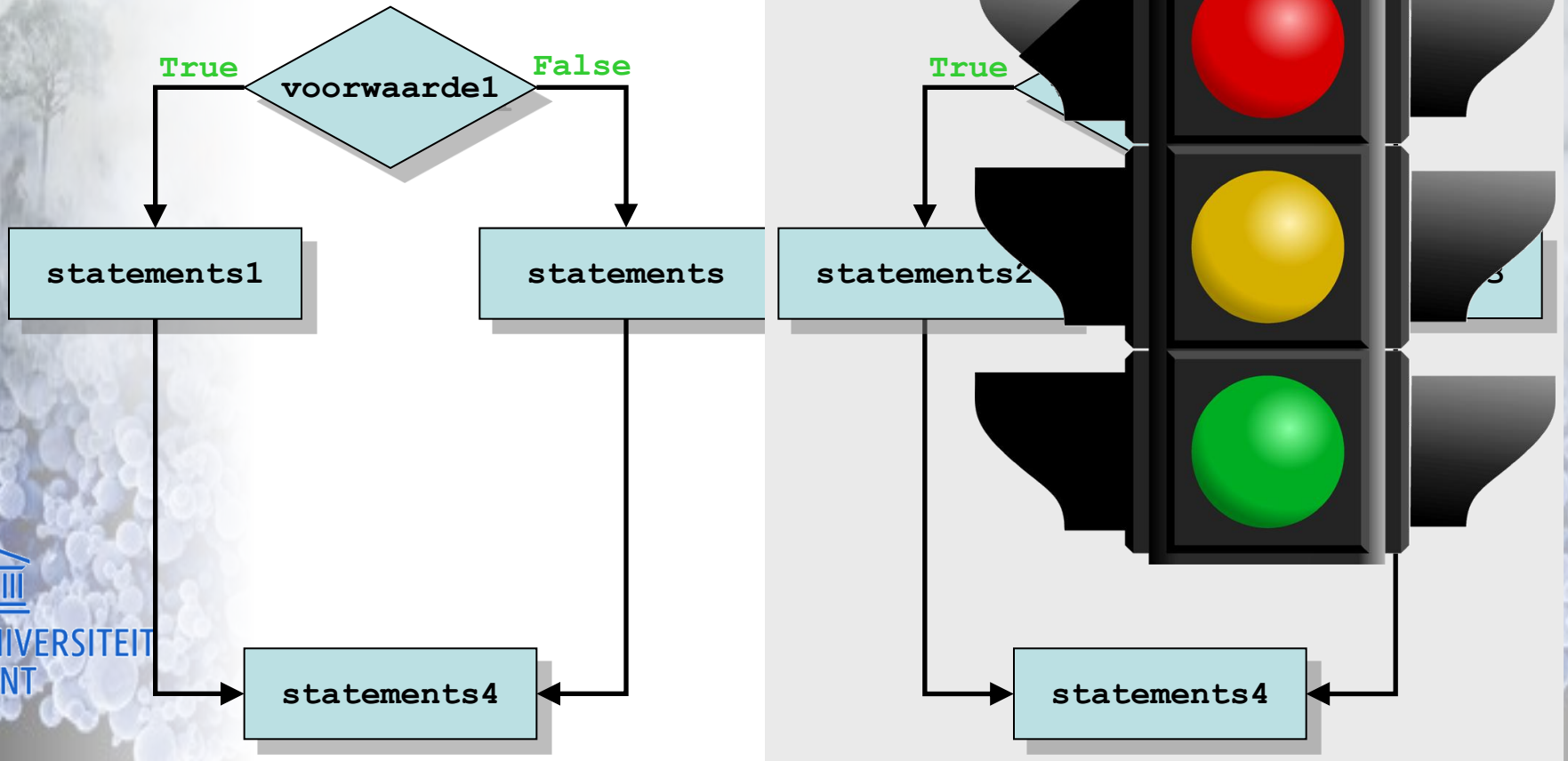
Geketende voorwaarden





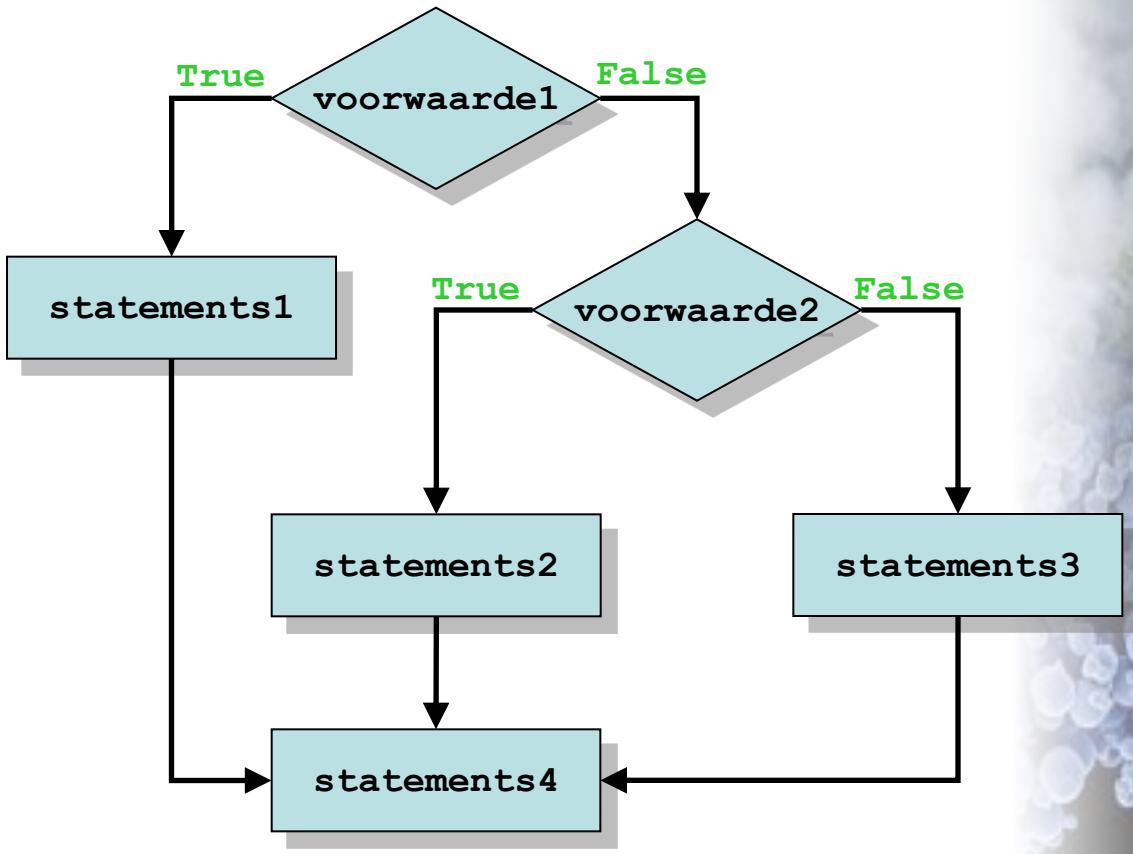
UNIVERSITEIT GENT

Geketende voorwaarden



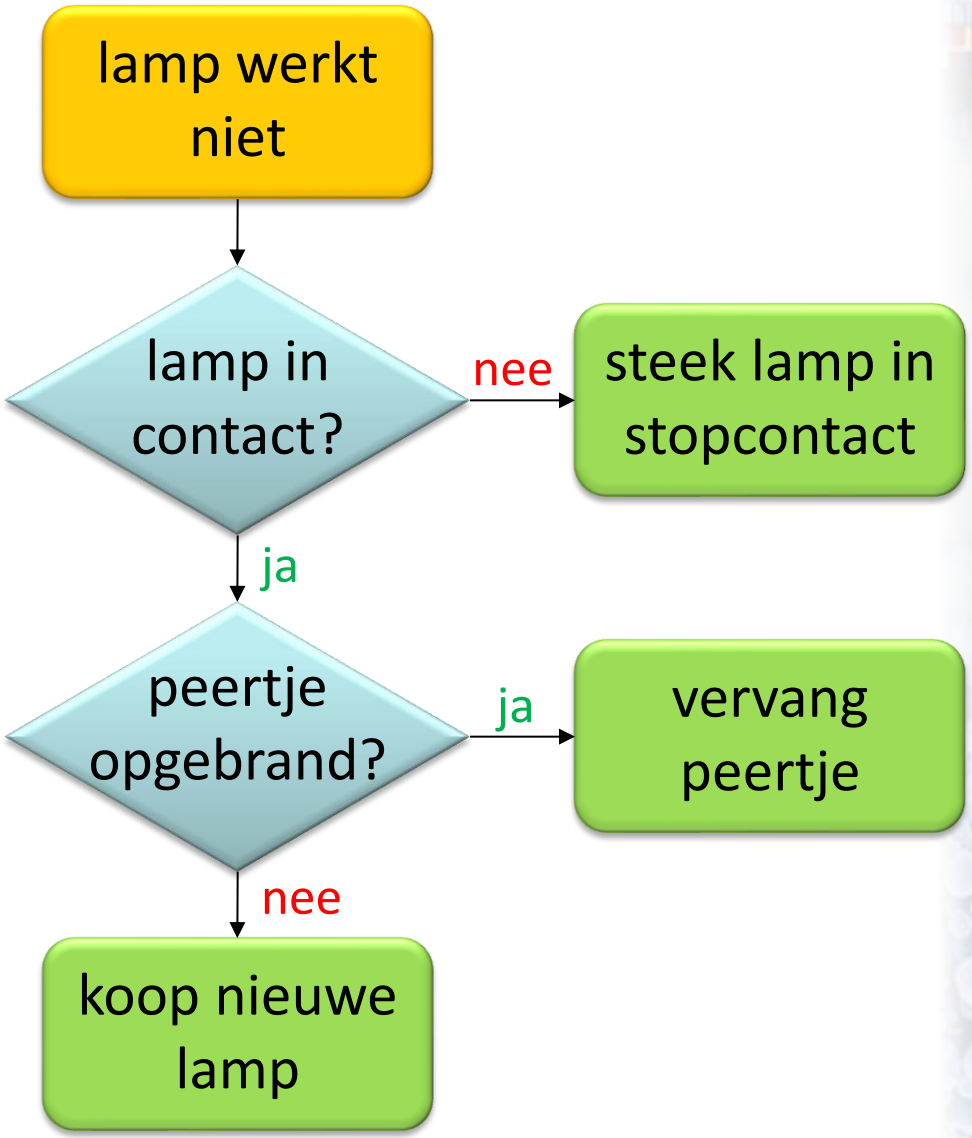
UNIVERSITEIT GENT

Geketende voorwaarden



UNIVERSITEIT GENT

Geketende voorwaarden

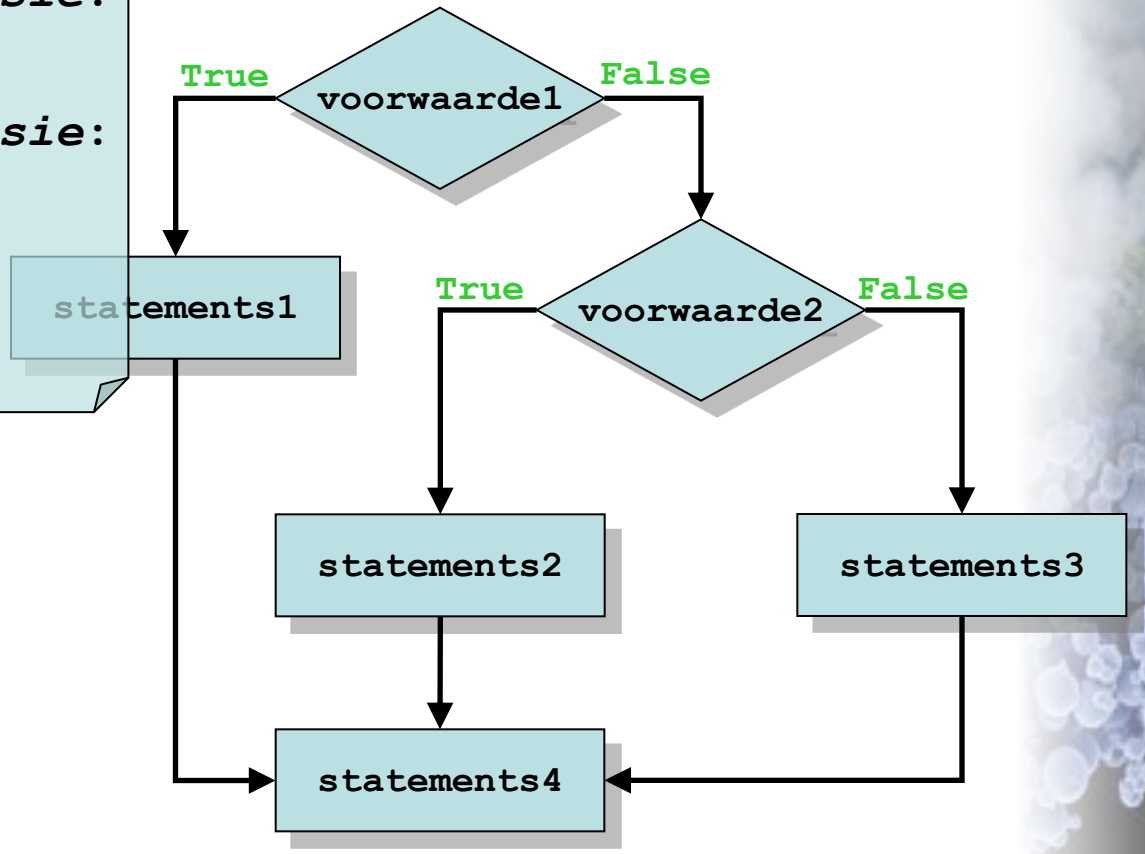


UNIVERSITEIT GENT

Geketende voorwaarden

syntaxis

```
if booleaanse expressie:
    statements
[elif booleaanse expressie:
    statements]
[elif booleaanse expressie:
    statements]
[else:
    statements]
```



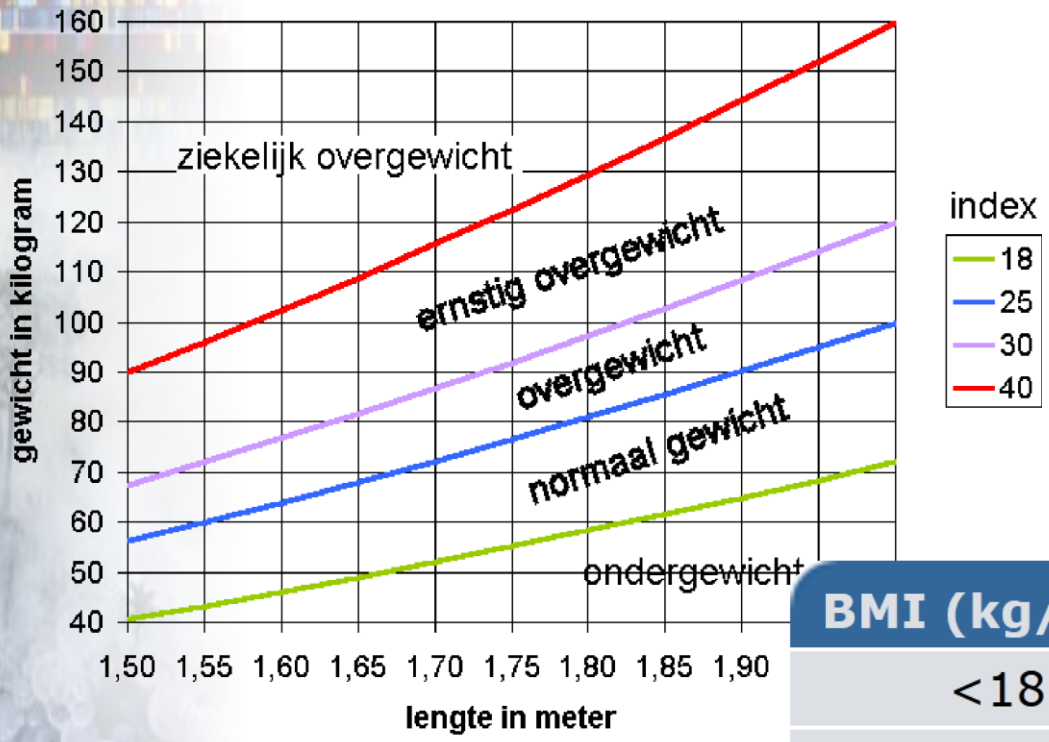
UNIVERSITEIT GENT

Geketende voorwaarden

```
>>> x = 3; y = 4
>>> if x < y:
...     print(f'{x} is kleiner dan {y}')
... elif x > y:
...     print(f'{x} is groter dan {y}')
... else:
...     print(f'{x} en {y} zijn gelijk')
...
3 is kleiner dan 4
```

UNIVERSITEIT GENT

BMI



$$\text{BMI} = \frac{\text{gewicht}}{\text{lengte}^2}$$

BMI (kg/m^2)	interpretatie
<18	ondergewicht
[18,25[	normaal gewicht
[25,27[	licht overgewicht
[27,30[	matig overgewicht
[30,40[	ernstig overgewicht
≥40	ziekelijk overgewicht

UNIVERSITEIT GENT

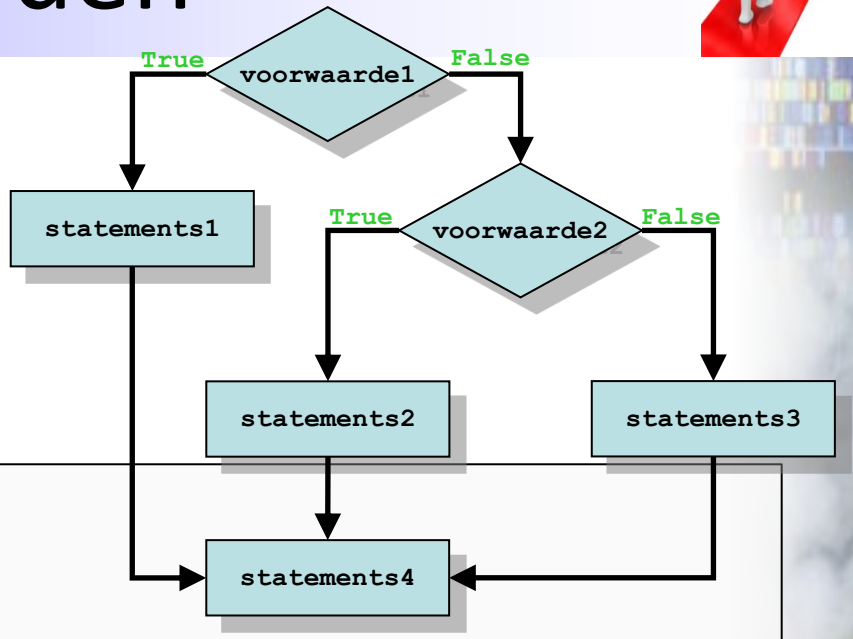
BMI

BMI.py

```
# persoonsgegevens inlezen
gewicht = int(input('Geef je gewicht in kilogram: '))
lengte = int(input('Geef je lengte in centimeter: '))

# BMI berekenen
BMI = gewicht / (lengte / 100) ** 2      # 2x reële deling

# BMI interpreteren
if BMI < 18:
    interpretatie = 'ondergewicht'
elif BMI < 25:
    interpretatie = 'normaal gewicht'
elif BMI < 27:
    interpretatie = 'licht overgewicht'
elif BMI < 30:
    interpretatie = 'matig overgewicht'
elif BMI < 40:
    interpretatie = 'ernstig overgewicht'
else:
    interpretatie = 'ziekelijk overgewicht'

# interpretatie uitschrijven
print('Een persoon van', gewicht, 'kg met een lengte van', lengte,
    'cm heeft', interpretatie + '.')
```

UNIVERSITEIT GENT

Geneste voorwaarden




```
>>> x = 3; y = 4
>>> if x == y:
...     print(f'{x} en {y} zijn gelijk')
... else:
...     if x < y:
...         print(f'{x} is kleiner dan {y}')
...     else:
...         print(f'{x} is groter dan {y}')
...
3 is kleiner dan 4
```

UNIVERSITEIT GENT

Geneste voorwaarden

- structuur wordt duidelijk aangegeven via insprongen
  - geneste voorwaarden zijn al snel moeilijk te lezen
  - goed idee om ze te proberen vermijden indien mogelijk

```
>>> x = 3; y = 4
>>> if x == y:
...     print(f'{x} en {y} zijn gelijk')
... else:
...     if x < y:
...         print(f'{x} is kleiner dan {y}')
...     else:
...         print(f'{x} is groter dan {y}')
...
3 is kleiner dan 4
```

UNIVERSITEIT GENT

Geneste voorwaarden

- structuur wordt duidelijk aangegeven via insprongen
  - geneste voorwaarden zijn al snel moeilijk te lezen
  - goed idee om ze te proberen vermijden indien mogelijk
    - hiervoor kunnen vaak logische operatoren gebruikt worden

```
if 0 <= x:
    if x < 10:
        print(f'{x} is positief getal van 1 cijfer.')
```

UNIVERSITEIT GENT

Geneste voorwaarden

- structuur wordt duidelijk aangegeven via insprongen
  - geneste voorwaarden zijn al snel moeilijk te lezen
  - goed idee om ze te proberen vermijden indien mogelijk
    - hiervoor kunnen vaak logische operatoren gebruikt worden

```
if 0 <= x and x < 10:
    print(f'{x} is positief getal van 1 cijfer.')
```

UNIVERSITEIT GENT

Geneste voorwaarden

- structuur wordt duidelijk aangegeven via insprongen
  - geneste voorwaarden zijn al snel moeilijk te lezen
  - goed idee om ze te proberen vermijden indien mogelijk
    - hiervoor kunnen vaak logische operatoren gebruikt worden

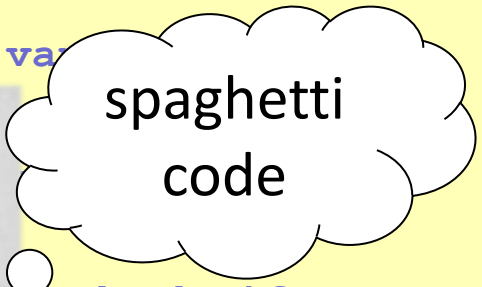
```
if 0 <= x < 10:
    print(f'{x} is positief getal van 1 cijfer.')
```

UNIVERSITEIT GENT

Python syntaxisregels

```
if voorwaarde1:                    # begin va
if voorwaarde
if voorwaarde
statements1
elif voorwaa
statements2
else:
statements3

else:
if voorwaarde
statements4


else:
statements5
```

```
                                   ij derde if
                                   ij derde if
                                   van derde if
                                   ij tweede if
                                   van vierde if
                                   van vierde if
                                   van tweede if
                                   ij eerste if
# einde van eerste if
```



UNIVERSITEIT GENT

Python syntaxisregels

```
if voorwaarde1:                    # begin van eerste if
    if voorwa                      van tweede if
        if vo                      van derde if
            s
        elif                       ij derde if
            s
        else                       ij derde if
            s
                                   van derde if
    else:                          ij tweede if
        if vo                      van vierde if
            s
                                   van vierde if
                                   van tweede if
else:                              ij eerste if
    statemen
                                   # einde van eerste if
```

UNIVERSITEIT GENT

Programmeren met stijl

stijlregel

Python Style Guide (PEP8) beveelt 4 spaties aan als insprong, en zeker geen tabs

Python syntaxisregels

```
if voorwaarde1:                    # begin van eerste if
    if voorwaarde2:                # begin van tweede if
        if voorwaarde3:            # begin van derde if
            statements1
        elif voorwaarde4:          # elif bij derde if
            statements2
        else:                      # else bij derde if
            statements3
                                   # einde van derde if
    else:                          # else bij tweede if
        if voorwaarde5:            # begin van vierde if
            statements4
                                   # einde van vierde if
                                   # einde van tweede if
else:                              # else bij eerste if
    statements5
                                   # einde van eerste if
```

UNIVERSITEIT GENT

Python syntaxisregels

```
if voorwaarde1:                  # begin van eerste if
    if voorwaarde2:              # begin van tweede if
        if voorwaarde3:          # begin van derde if
            statements1
        elif voorwaarde4:        # elif bij derde if
            statements2
        else:                    # else bij derde if
            statements3
    else:                        # else bij tweede if
        if voorwaarde5:          # begin van vierde if
            statements4
else:                            # else bij eerste if
    statements5
```

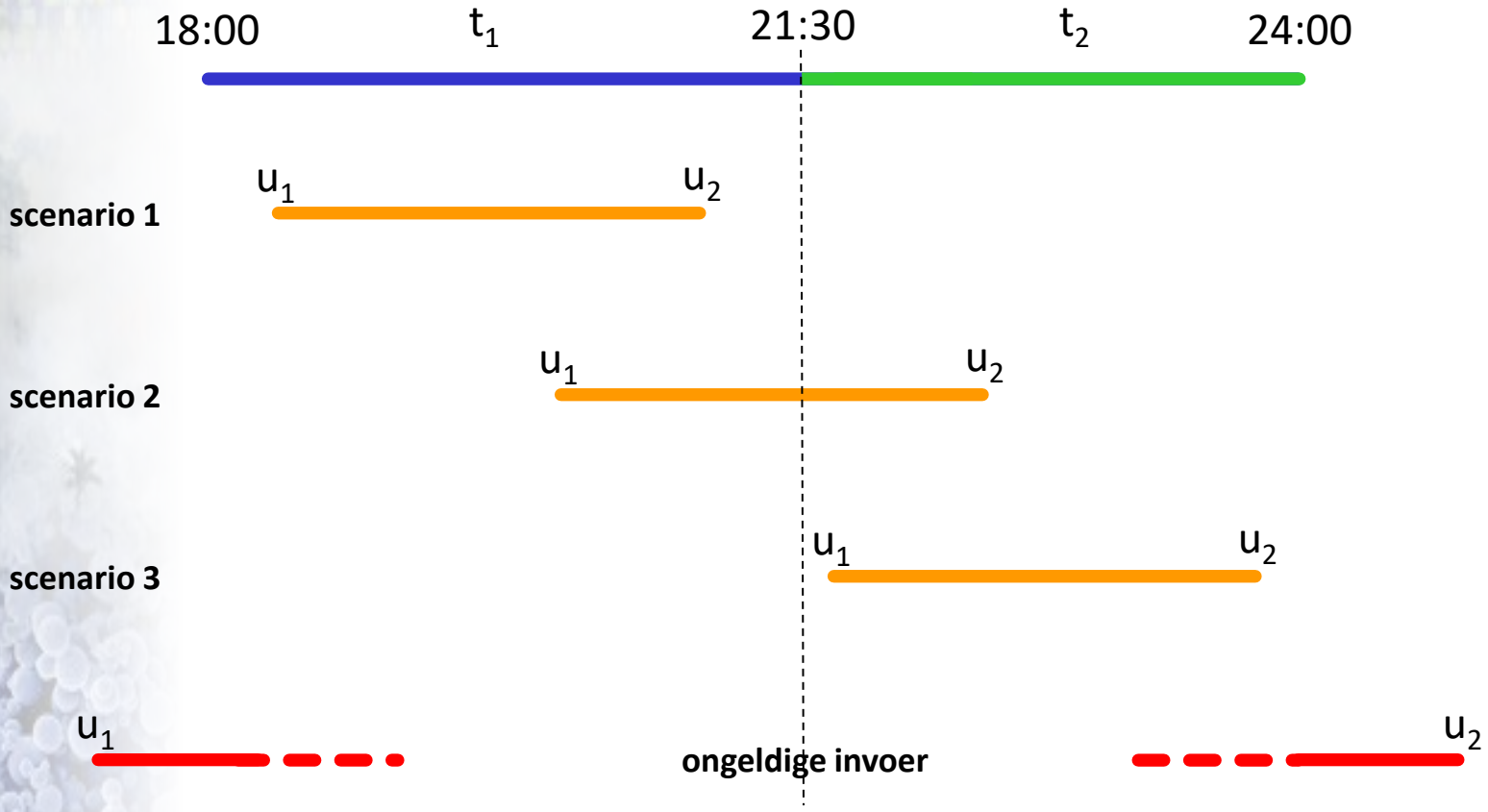
UNIVERSITEIT GENT

Babysit

opgave

Een babysitter rekent €2 per uur tussen 18:00 en 21:30, en €4 per uur tussen 21:30 en middernacht. Ze wil niet babysitten vóór 18:00 en ook niet na middernacht.

Babysit



UNIVERSITEIT GENT

Babysit

babysit1.py

```
# tijdstippen inlezen
u1 = int(input())
m1 = int(input())
u2 = int(input())
m2 = int(input())

# tijdstippen omzetten naar uren (sinds middernacht)
u1 = u1 + m1 / 60            # let op: reële deling
u2 = u2 + m2 / 60
u1800 = 18.0                 # 18:00 in uren sinds middernacht
u2130 = 21.5                 # 21:30 in uren sinds middernacht

if u1 < u1800 or u2 < u1:
    print('ongeldige invoer')
else:
    # bereken aantal uren gebabysit voor (t1) en na (t2) 21:30
    if u2 < u2130:           # scenario 1
        t1 = u2 - u1
        t2 = 0.0
    elif u1 < u2130:         # scenario 2
        t1 = u2130 - u1
        t2 = u2 - u2130
    else:                    # scenario 3
        t2 = u2 - u1
        t1 = 0.0

    # bereken totaal verdiende bedrag
    print(2 * t1 + 4 * t2)
```

UNIVERSITEIT GENT

Babysit

babysit2.py


```
# tijdstippen inlezen
u1 = int(input())
m1 = int(input())
u2 = int(input())
m2 = int(input())

# tijdstippen omzetten naar uren (sinds middernacht)
u1 = u1 + m1 / 60            # let op: reële deling
u2 = u2 + m2 / 60
u1800 = 18.0                 # 18:00 in uren sinds middernacht
u2130 = 21.5                 # 21:30 in uren sinds middernacht

if u1 < u1800 or u2 < u1:
    print('ongeldige invoer')
else:
    # bereken aantal uren gebabysit voor (t1) en na (t2) 21:30
    t1 = max(0, min(u2130, u2) - u1)
    t2 = max(0, u2 - max(u2130, u1))

    # bereken totaal verdiende bedrag
    print(2 * t1 + 4 * t2)
```

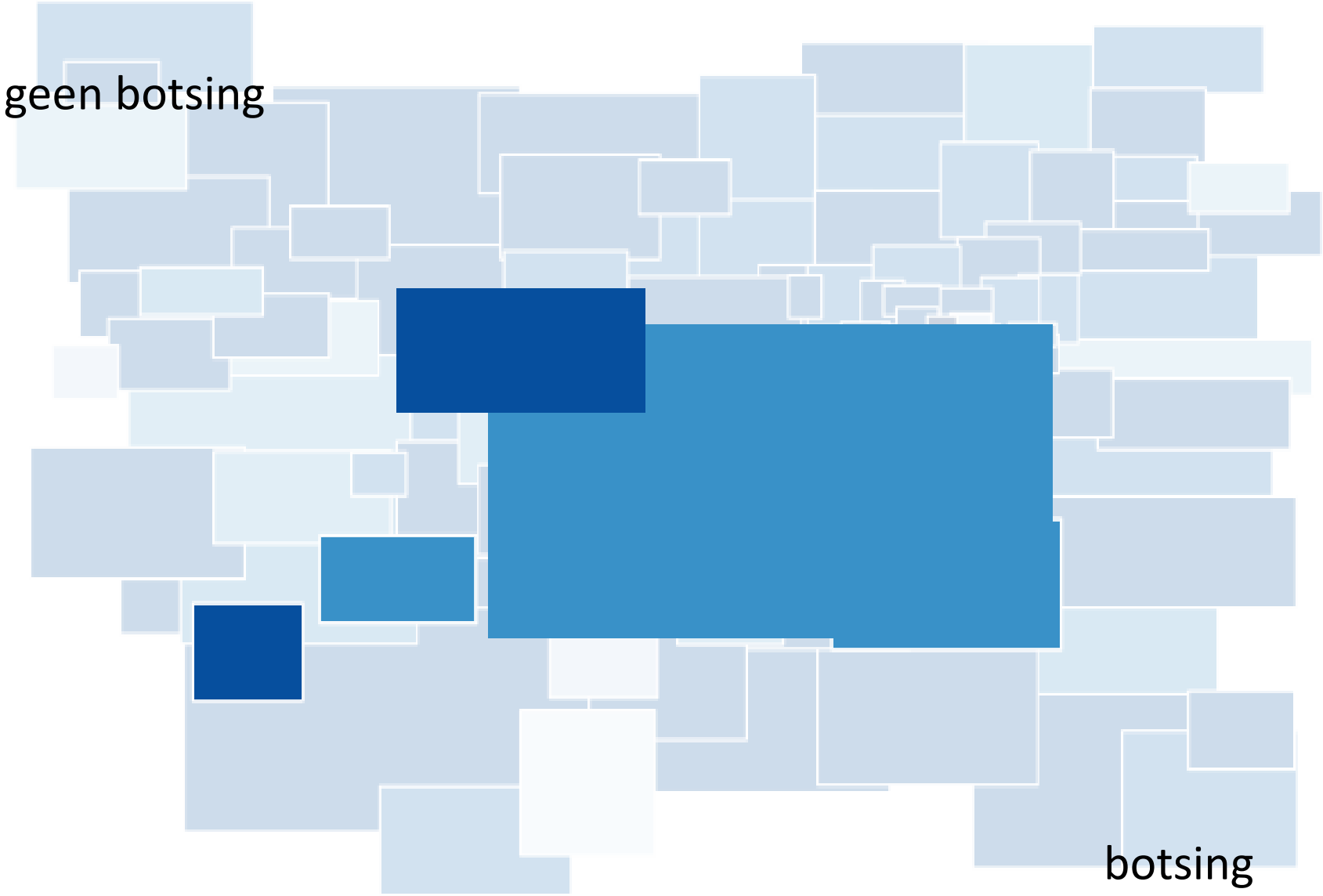
UNIVERSITEIT GENT

Botsing

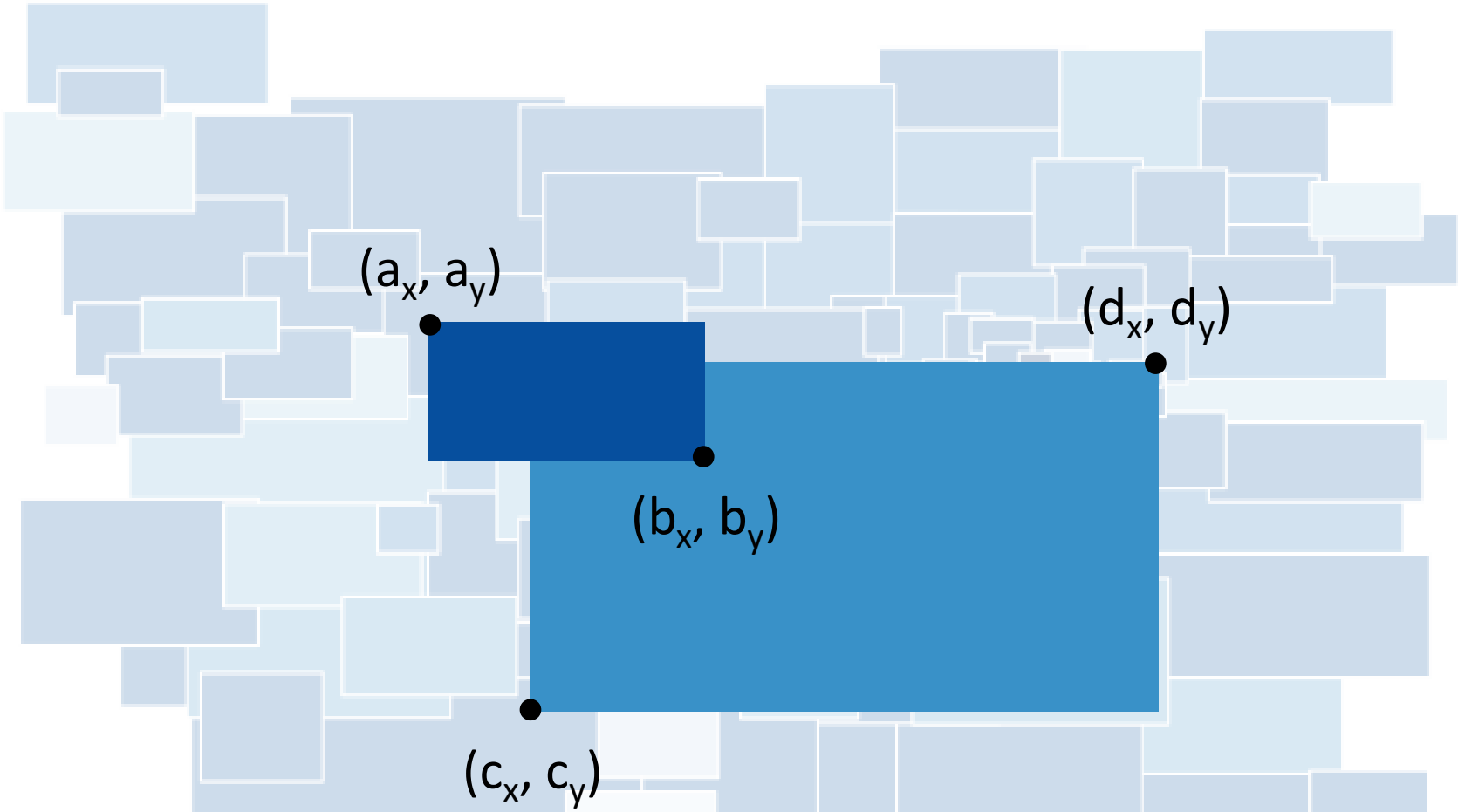
opgave

Bepaal of twee rechthoeken
al dan niet overlappen.

Botsing

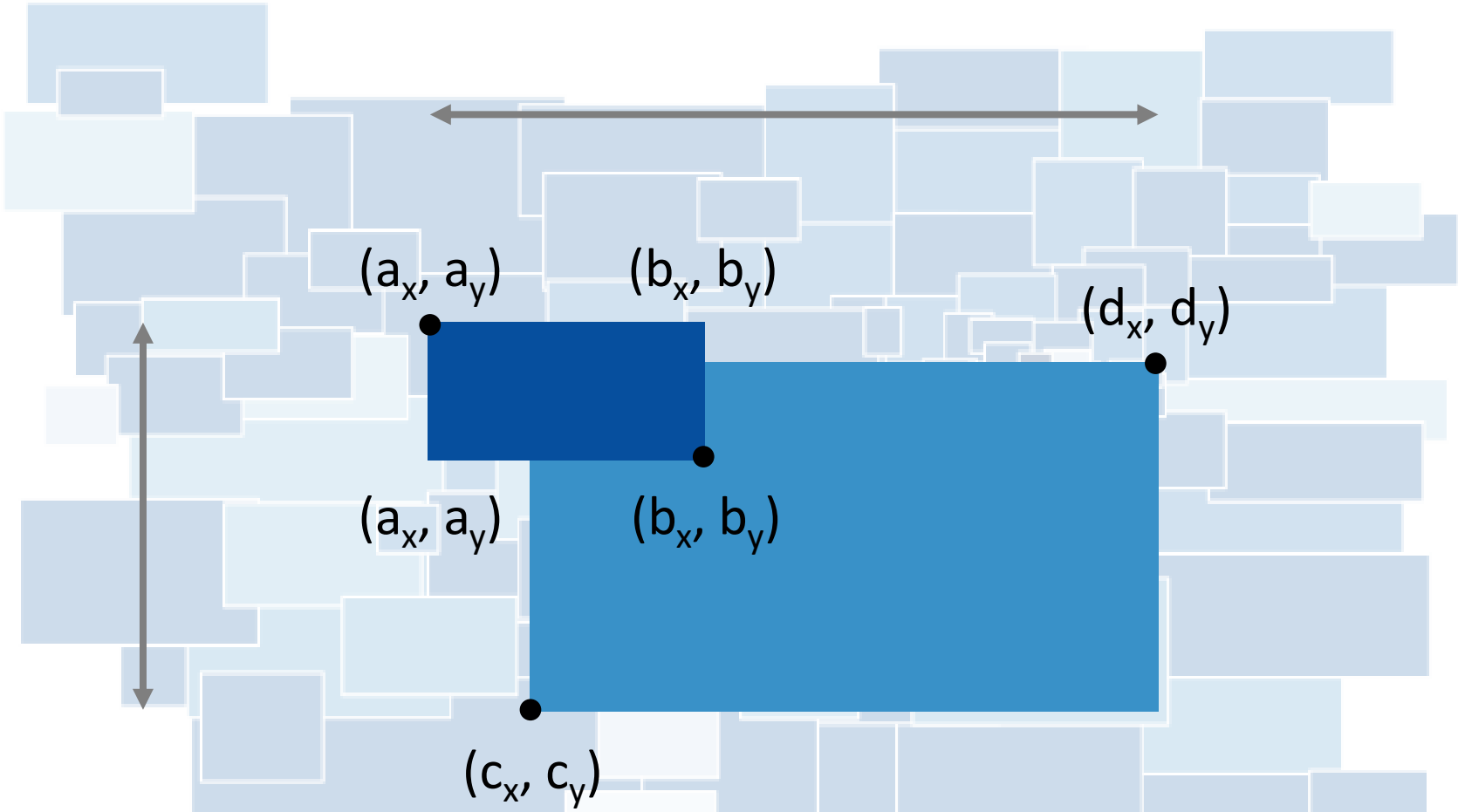


Botsing




```
ax, bx = min(ax, bx), max(ax, bx)
```

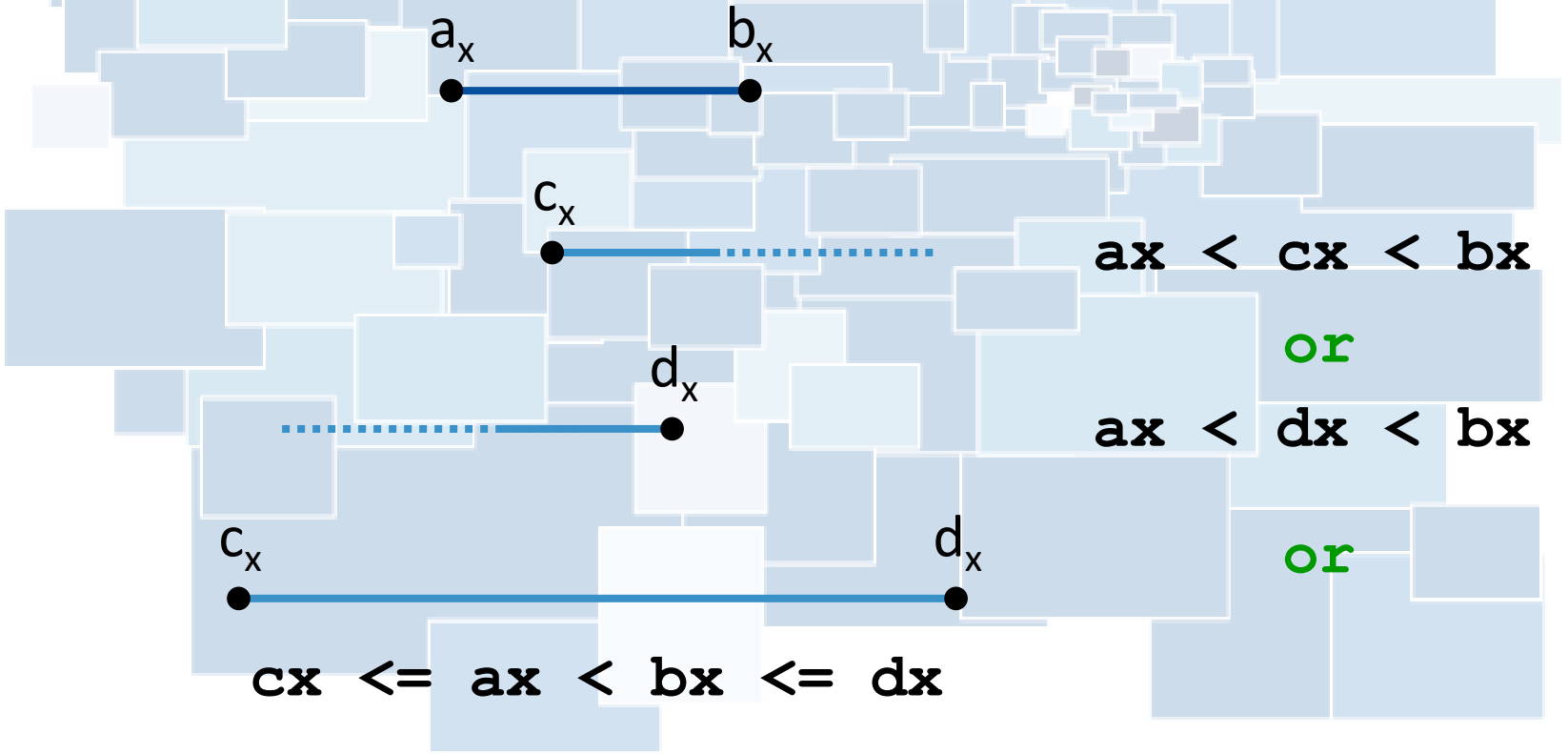
Botsing




```
ax, bx = min(ax, bx), max(ax, bx)
```

Botsing

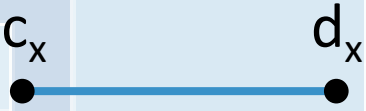
Wanneer overlapt dit lijnstuk met het lijnstuk c_x,d_x ?



Botsing

Wanneer overlapt dit **niet** lijnstuk met het lijnstuk c_x,d_x ?


```
dx <= ax
  or
bx <= cx
```




```
not(dx<ax and bx<cx)
```

Botsing

botsing.py

```
# diametraal overstaande punten van twee rechthoeken inlezen
ax, ay = int(input()), int(input())
bx, by = int(input()), int(input())
cx, cy = int(input()), int(input())
dx, dy = int(input()), int(input())

# neem telkens linkeronderhoek en rechterbovenhoek
ax, bx = min(ax, bx), max(ax, bx)
ay, by = min(ay, by), max(ay, by)
cx, dx = min(cx, dx), max(cx, dx)
cy, dy = min(cy, dy), max(cy, dy)

# nagaan of rechthoeken horizontaal en verticaal overlappen
horizontaal = ax < dx and cx < bx
verticaal = ay < dy and cy < by

# nagaan of rechthoeken al dan niet overlappen
if horizontaal and verticaal:
    print('botsing')
else:
    print('geen botsing')
```

UNIVERSITEIT GENT

Botsing

botsing.py

```
# diametraal overstaande punten van twee rechthoeken inlezen
ax, ay = int(input()), int(input())
bx, by = int(input()), int(input())
cx, cy = int(input()), int(input())
dx, dy = int(input()), int(input())

# neem telkens linkeronderhoek en rechterbovenhoek
ax, bx = min(ax, bx), max(ax, bx)
ay, by = min(ay, by), max(ay, by)
cx, dx = min(cx, dx), max(cx, dx)
cy, dy = min(cy, dy), max(cy, dy)

# nagaan of rechthoeken horizontaal en verticaal overlappen
horizontaal = ax < dx and cx < bx
verticaal = ay < dy and cy < by

# nagaan of rechthoeken al dan niet overlappen
print('botsing' if horizontaal and verticaal else 'geen botsing')
```

UNIVERSITEIT GENT

Huiswerk (volgende les)

- handboek: lees hoofdstuk 2 (controlestructuren)
- bekijk videohandleidingen
  - broncode debuggen
  - werken met floating point getallen
  - BMI (uitgewerkt voorbeeld)
  - Babysit (uitgewerkt voorbeeld)




UNIVI GENT

Huiswerk (werkcolleges)

- opgelegde oefeningen reeks 02 afwerken (deadline dinsdag 7 oktober 2025, 22:00)
  - ISBN (video)
  - Reynoldsgetal
  - Blad, steen en schaar
  - Seizoenen
  - Trilateratie
  - Darts

Vragen of opmerkingen?

UNIVERSITEIT GENT

The sky is the limit …

"What I hear I forget.
What I see I remember.
What I do I understand."

— Confucius

UNIVERSITEIT GENT