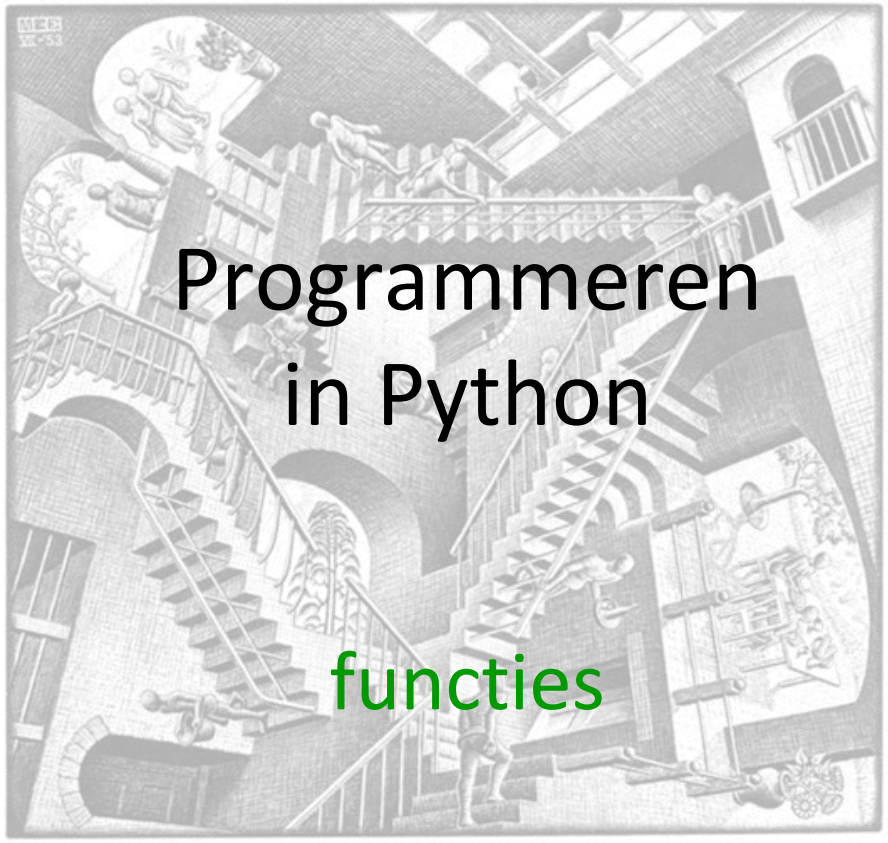
“Civilization advances by extending the number of important operations which we can perform without thinking about them.”

— Alfred North Whitehead

Universiteit Gent
Faculteit Wetenschappen



Programmeren in Python

functies

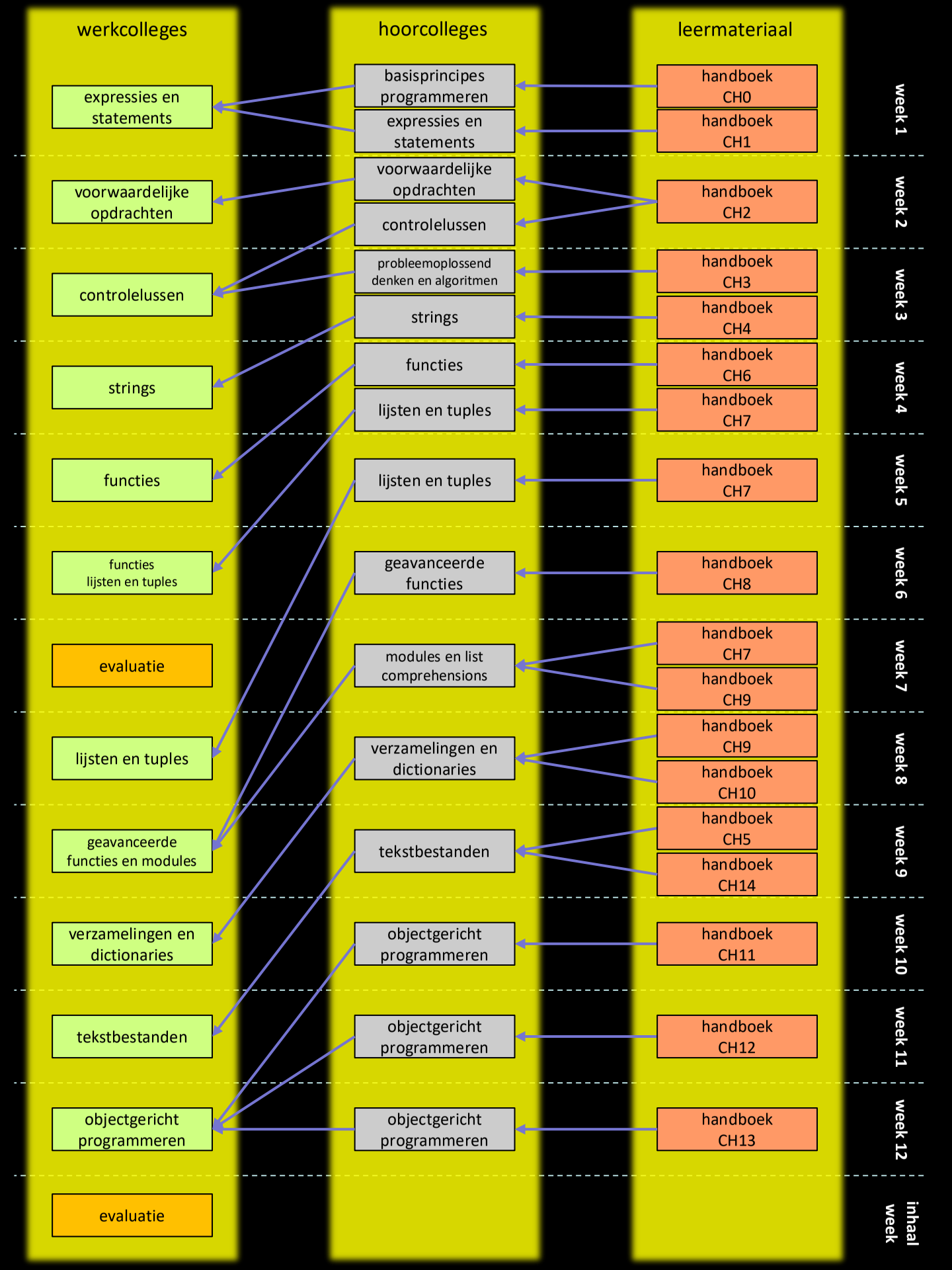
prof. dr. Peter Dawyndt

peter.dawyndt@ugent.be

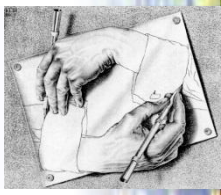
@dawyndt

UNIVERSITEIT GENT

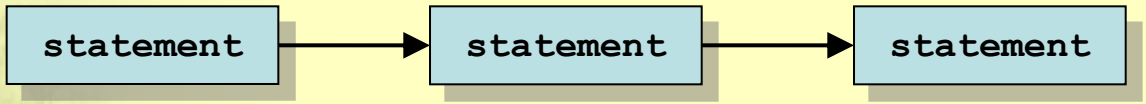
6

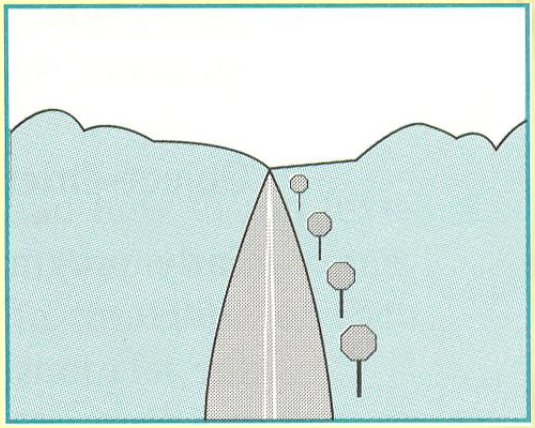

werkcolleges
hoorcolleges
leermateriaal
expressies en statements
voorwaardelijke opdrachten
controlelussen
strings
functies
functies lijsten en tuples
evaluatie
lijsten en tuples
geavanceerde functies en modules
verzamelingen en dictionaries
tekstbestanden
objectgericht programmeren
evaluatie
basisprincipes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

Controlestructuren

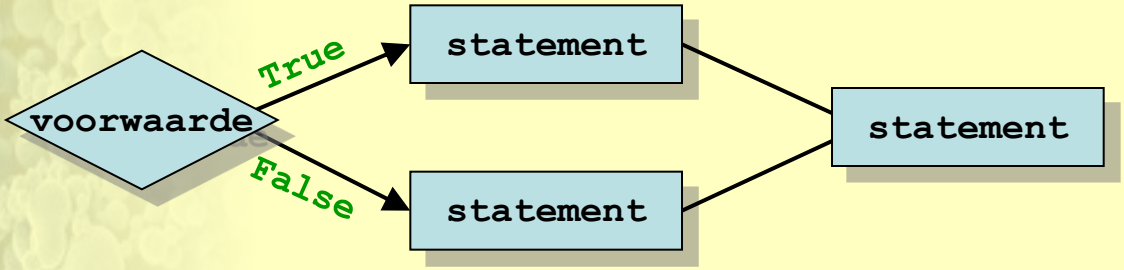


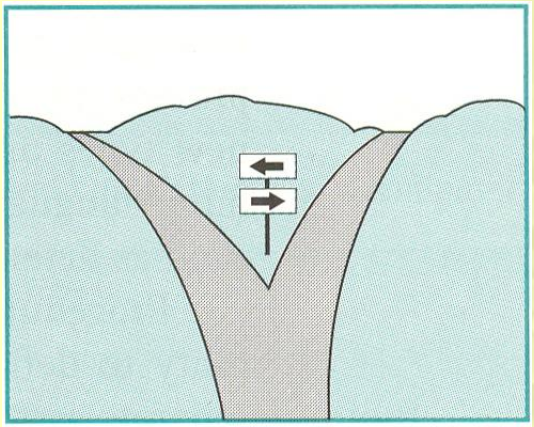
sequentie



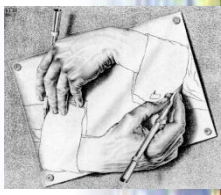


voorwaardelijke opdracht (selectie, sprong)

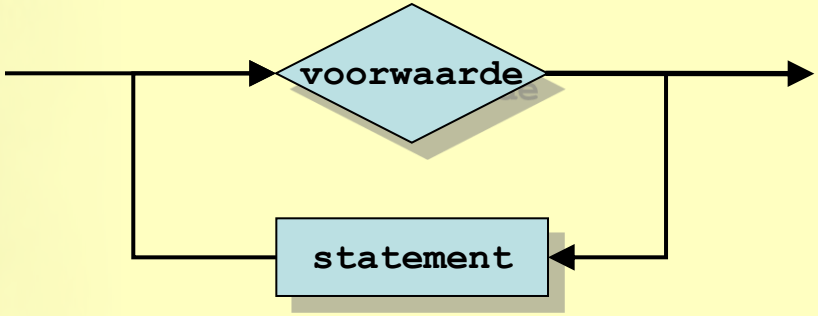


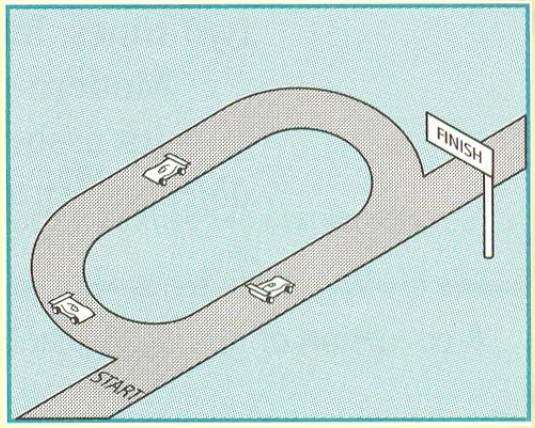


Controlestructuren

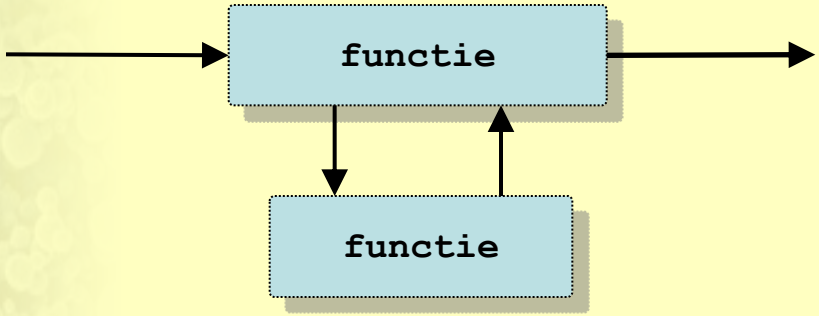


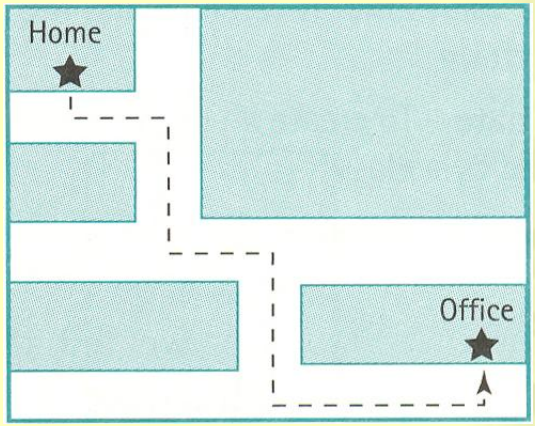
repetitieve opdracht (lus, herhaling, iteratie)





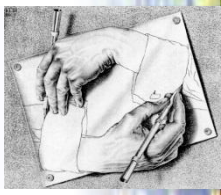
functie (methode)






I THINK YOU SHOULD BE MORE SPECIFIC HERE IN STEP TWO
THEN A MIRACLE OCCURS
S. Harris

Parameters & argumenten

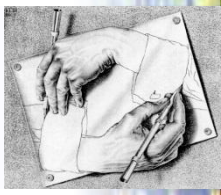


- **argument**: object (expressie) dat wordt meegegeven wanneer functie wordt aangeroepen
  - object toegekend aan corresponderende parameter van functie
- **parameter**: naam die binnen een functie gebruikt wordt om te verwijzen naar object dat als argument werd doorgegeven

```
>>> abs(5)
5
>>> abs(-5)
5
>>> pow(2, 3)
8
>>> pow(7, 4)
2401
```

UNIVERSITEIT GENT

Parameters & argumenten

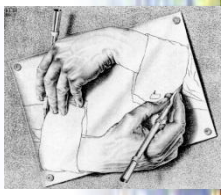


- **argument**: object (expressie) dat wordt meegegeven wanneer functie wordt aangeroepen
  - object toegekend aan corresponderende parameter van functie
- **parameter**: naam die binnen een functie gebruikt wordt om te verwijzen naar object dat als argument werd doorgegeven

```
>>> max(7, 11)
11
>>> max(4, 1, 17, 2, 12)
17
>>> max(3 * 11, 5 ** 3, 512 - 9, 1024 ** 0)
503
>>> max(3 * 11, pow(5, 3), 512 - 9, pow(1024, 0))
503
```

UNIVERSITEIT GENT

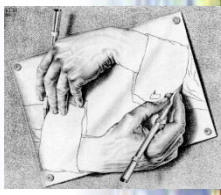
Ontwerpsfilosofie



- programmeertaal moet *niet* elke functionaliteit voorzien die iemand ooit zou willen hebben
- in plaats daarvan moet ze manieren voorzien om extra functionaliteit toe te voegen voor specifieke toepassingen
  - definieer functies om opdrachten op hoger niveau te kunnen uitvoeren (*abstraction*)

UNIVERSITEIT GENT

Ontwerpsfilosofie

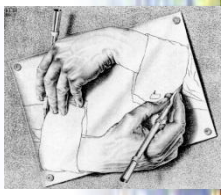


- programmeertaal moet *niet* elke functionaliteit voorzien die iemand ooit zou willen hebben
- in plaats daarvan moet ze manieren voorzien om extra functionaliteit toe te voegen voor specifieke toepassingen
  - definieer functies om opdrachten op hoger niveau te kunnen uitvoeren (*abstraction*)

"Maak een taal waarin de oplossing voor je originele probleem triviaal wordt."

UNIVERSITEIT GENT

Functies definiëren



- sleutelwoord **`def`** definieert een functie
  - namen van parameters staan tussen ronde haakjes
    - functiedefinitie: nul of meer parameters
  - legt gegevenstype van parameters en resultaat niet vast

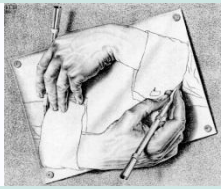
```
>>> def verdubbel(x):
...     return 2 * x
...
>>> verdubbel(12)
24
>>> verdubbel('lava')
'lavalava'
>>> verdubbel([1, 2, 3])
[1, 2, 3, 1, 2, 3]
```

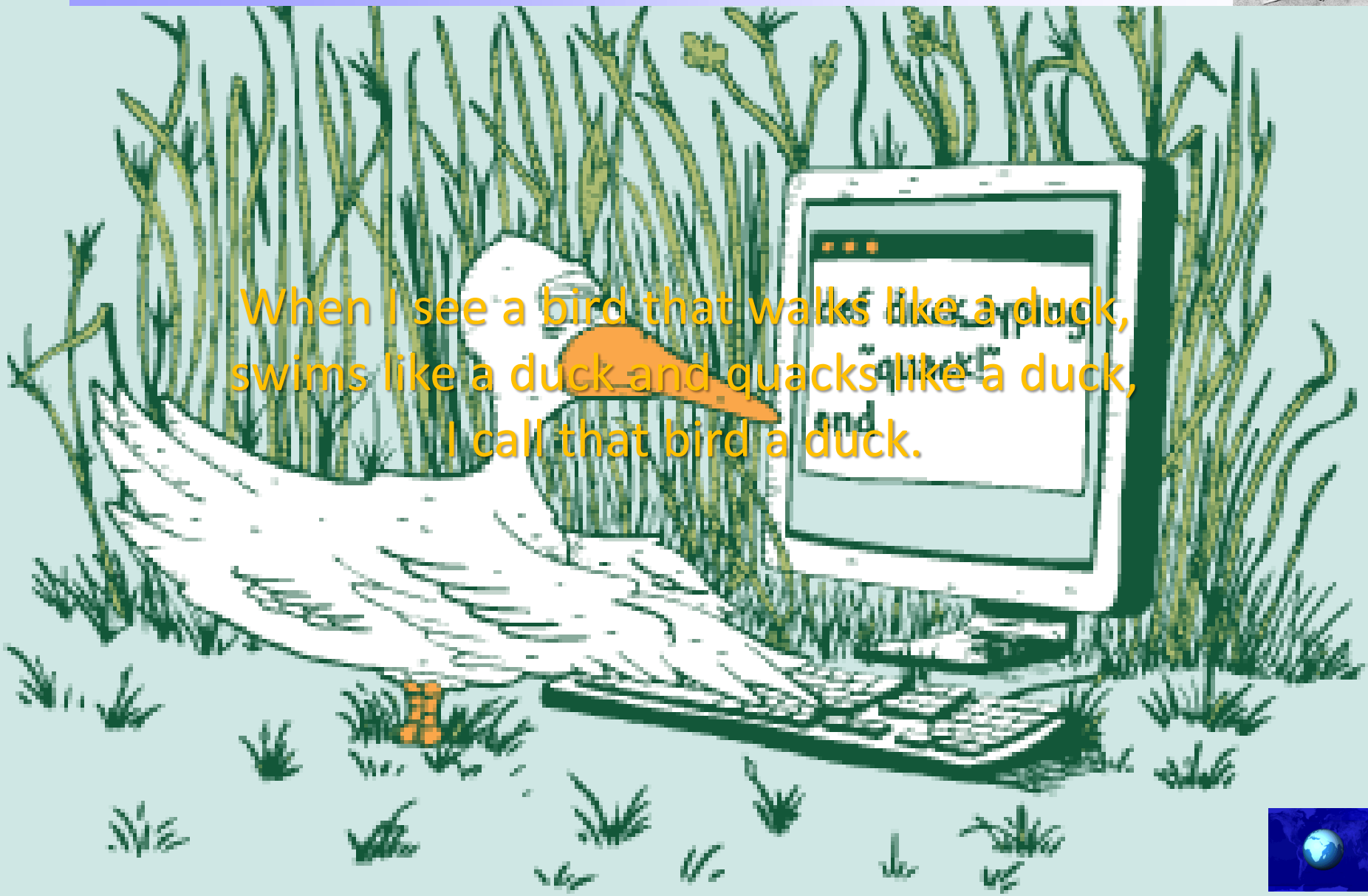
syntaxis

```
def naam([parameter, …]):
    statements
```

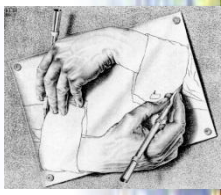
UNIVERSITEIT GENT

Duck typing



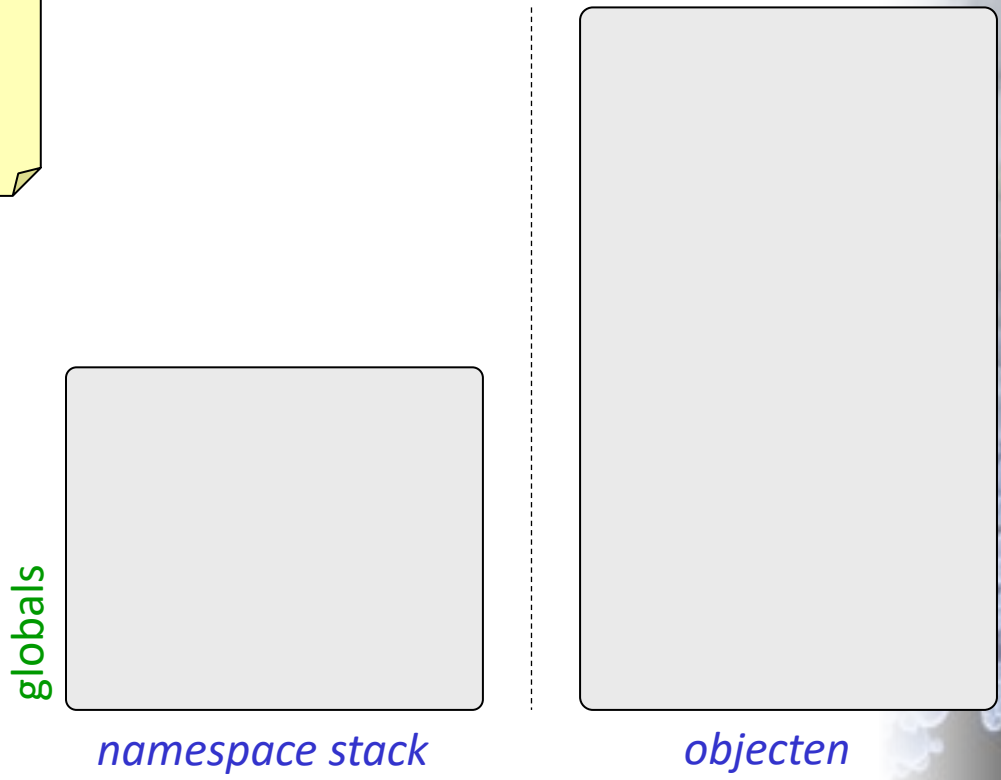

When I see a bird that walks like a duck,
swims like a duck and quacks like a duck,
I call that bird a duck.

Argumenten doorgeven



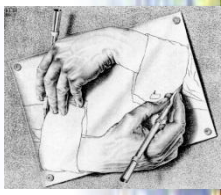
steensoort.py

```
def steensoort(naam):
    if naam in ['basalt', 'graniet']:
        soort = 'stollingsgesteente'
    elif naam in ['zandsteen', 'schalie']:
        soort = 'sedimentair gesteente'
    else:
        soort = 'metamorf gesteente'
    return soort

steen = 'zandsteen'
resultaat = steensoort(steen)
```



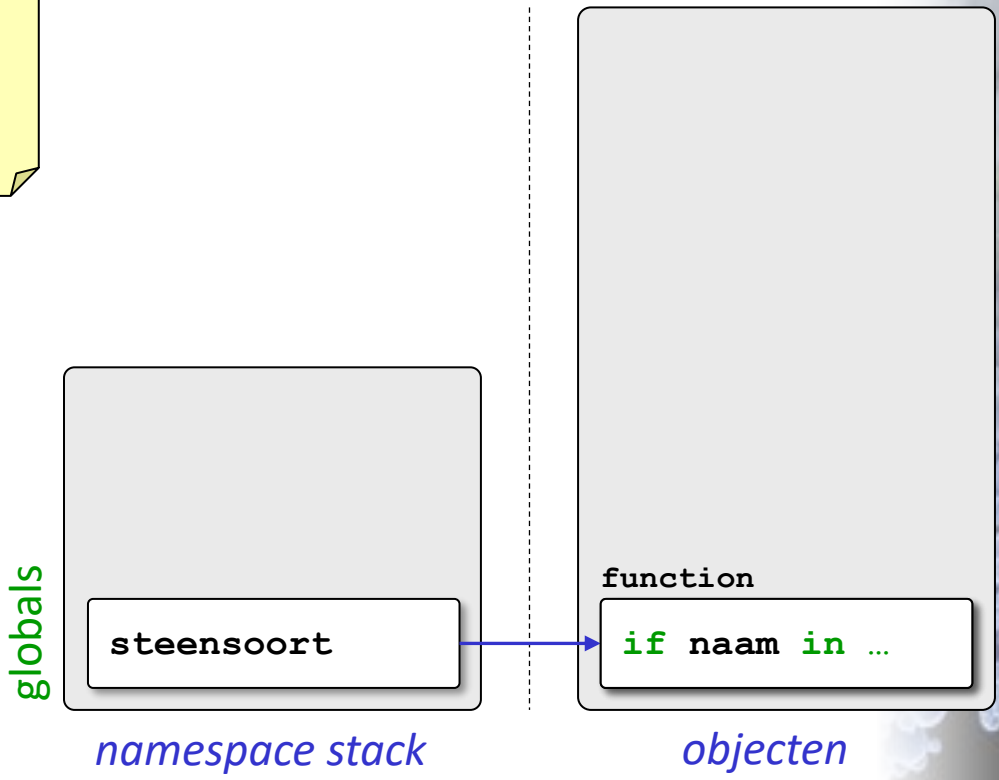
UNIVERSITEIT GENT

Argumenten doorgeven



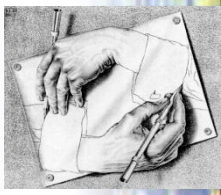
steensoort.py

```
def steensoort(naam):
    if naam in ['basalt', 'graniet']:
        soort = 'stollingsgesteente'
    elif naam in ['zandsteen', 'schalie']:
        soort = 'sedimentair gesteente'
    else:
        soort = 'metamorf gesteente'
    return soort

steen = 'zandsteen'
resultaat = steensoort(steen)
```



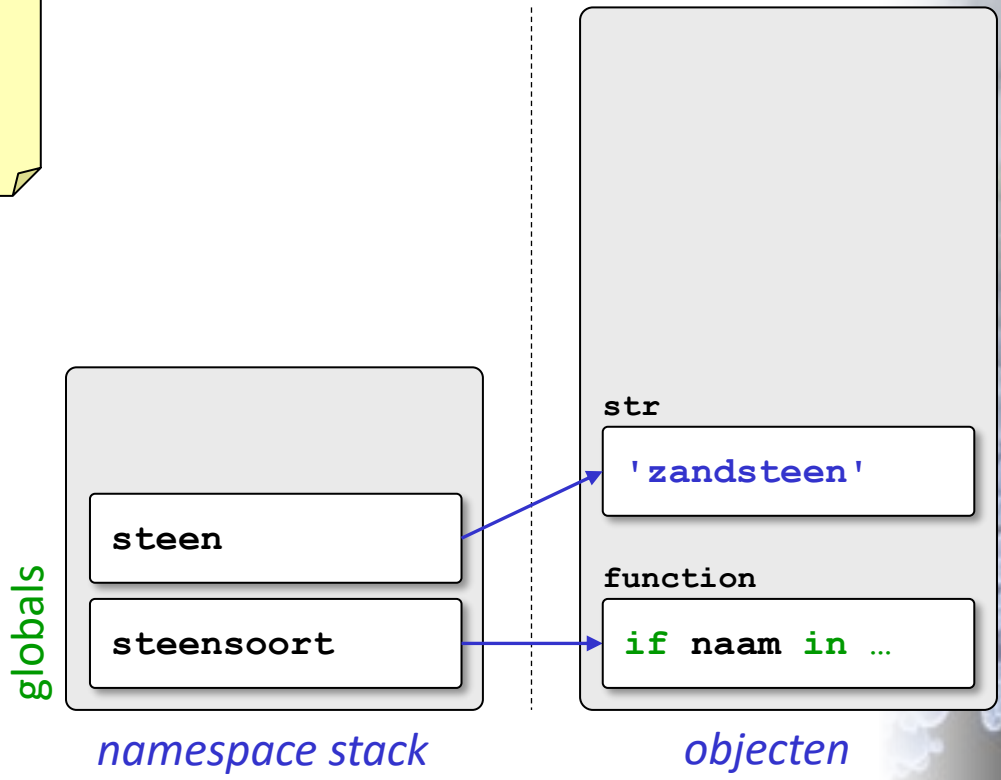
UNIVERSITEIT GENT

Argumenten doorgeven



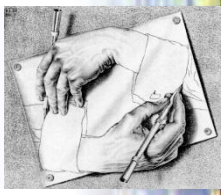
steensoort.py

```
def steensoort(naam):
    if naam in ['basalt', 'graniet']:
        soort = 'stollingsgesteente'
    elif naam in ['zandsteen', 'schalie']:
        soort = 'sedimentair gesteente'
    else:
        soort = 'metamorf gesteente'
    return soort

steen = 'zandsteen'
resultaat = steensoort(steen)
```



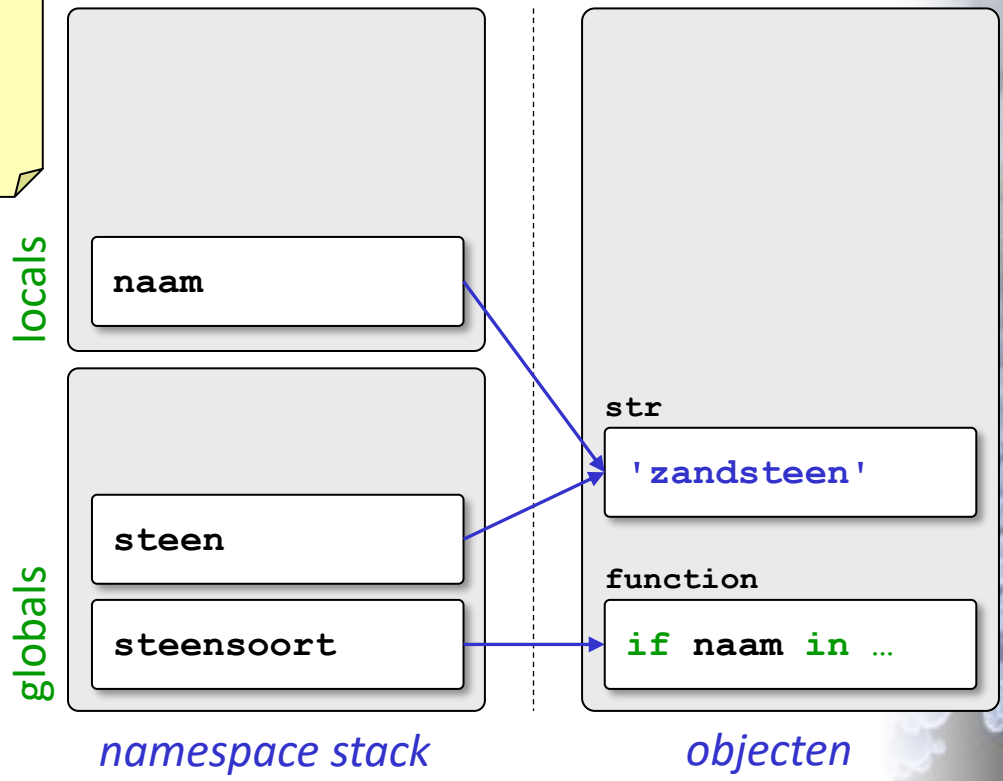
UNIVERSITEIT GENT

Argumenten doorgeven



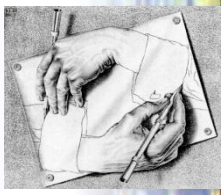
steensoort.py

```
def steensoort(naam):
    if naam in ['basalt', 'graniet']:
        soort = 'stollingsgesteente'
    elif naam in ['zandsteen', 'schalie']:
        soort = 'sedimentair gesteente'
    else:
        soort = 'metamorf gesteente'
    return soort

steen = 'zandsteen'
resultaat = steensoort(steen)
```



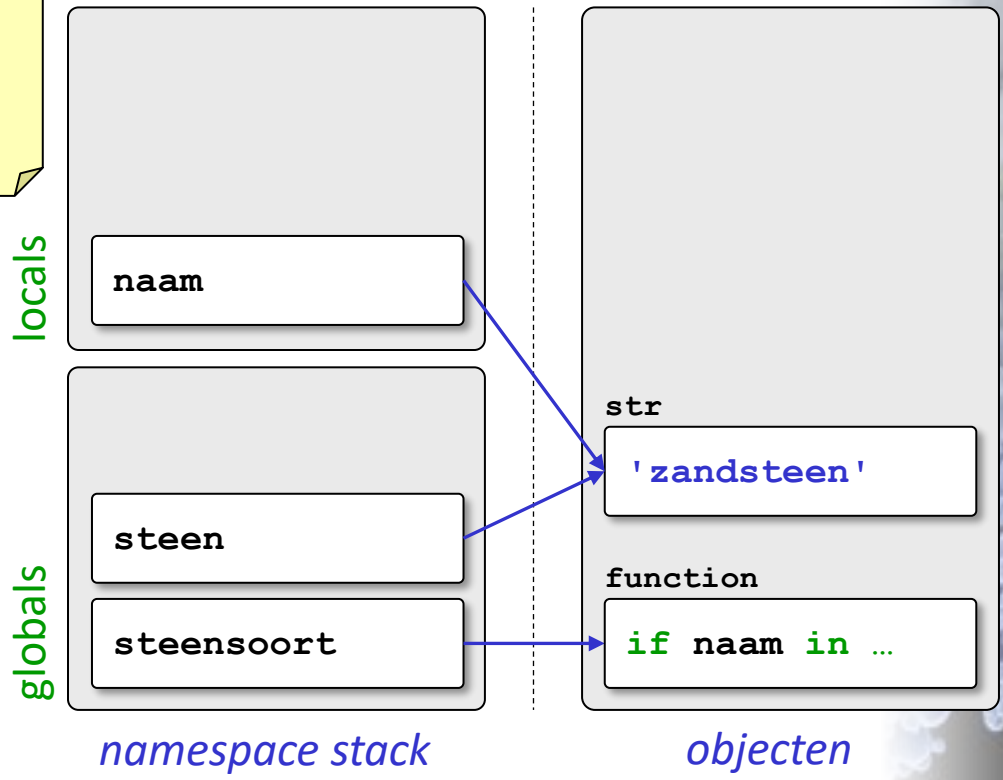
UNIVERSITEIT GENT

Argumenten doorgeven



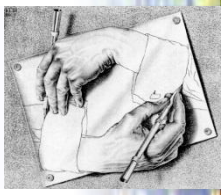
steensoort.py

```
def steensoort(naam):
    if naam in ['basalt', 'graniet']:
        soort = 'stollingsgesteente'
    elif naam in ['zandsteen', 'schalie']:
        soort = 'sedimentair gesteente'
    else:
        soort = 'metamorf gesteente'
    return soort

steen = 'zandsteen'
resultaat = steensoort(steen)
```



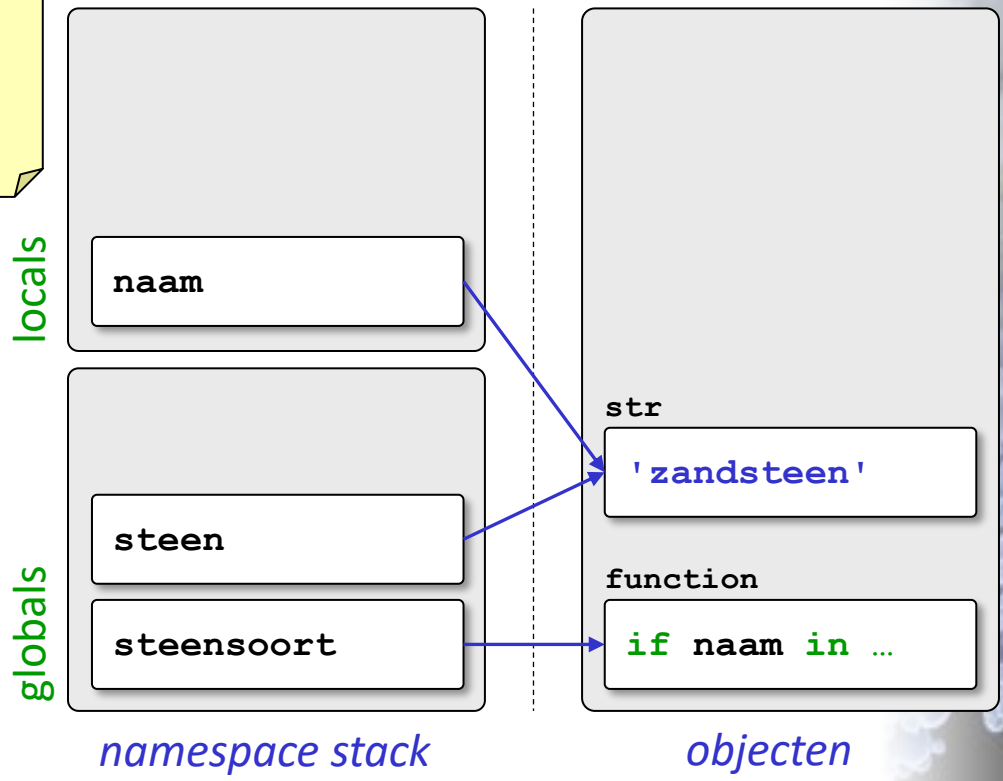
UNIVERSITEIT GENT

Argumenten doorgeven



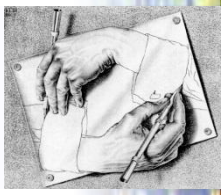
steensoort.py

```
def steensoort(naam):
    if naam in ['basalt', 'graniet']:
        soort = 'stollingsgesteente'
    elif naam in ['zandsteen', 'schalie']:
        soort = 'sedimentair gesteente'
    else:
        soort = 'metamorf gesteente'
    return soort

steen = 'zandsteen'
resultaat = steensoort(steen)
```



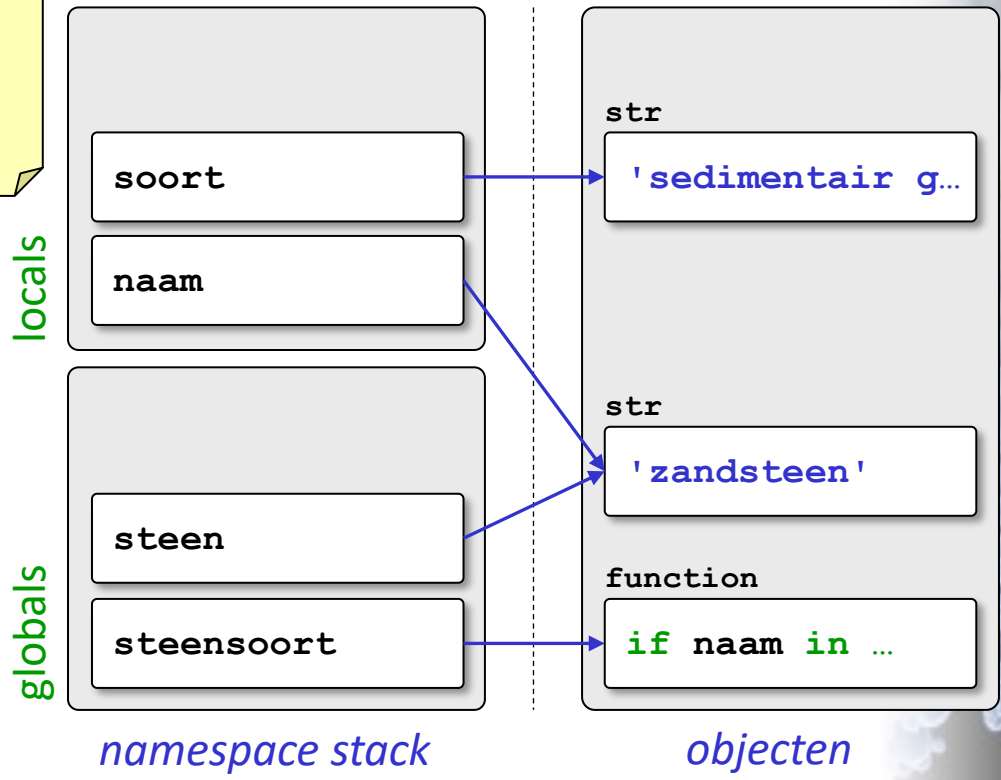
UNIVERSITEIT GENT

Argumenten doorgeven



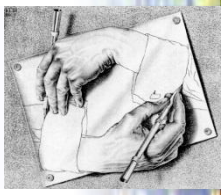
steensoort.py

```
def steensoort(naam):
    if naam in ['basalt', 'graniet']:
        soort = 'stollingsgesteente'
    elif naam in ['zandsteen', 'schalie']:
        soort = 'sedimentair gesteente'
    else:
        soort = 'metamorf gesteente'
    return soort

steen = 'zandsteen'
resultaat = steensoort(steen)
```



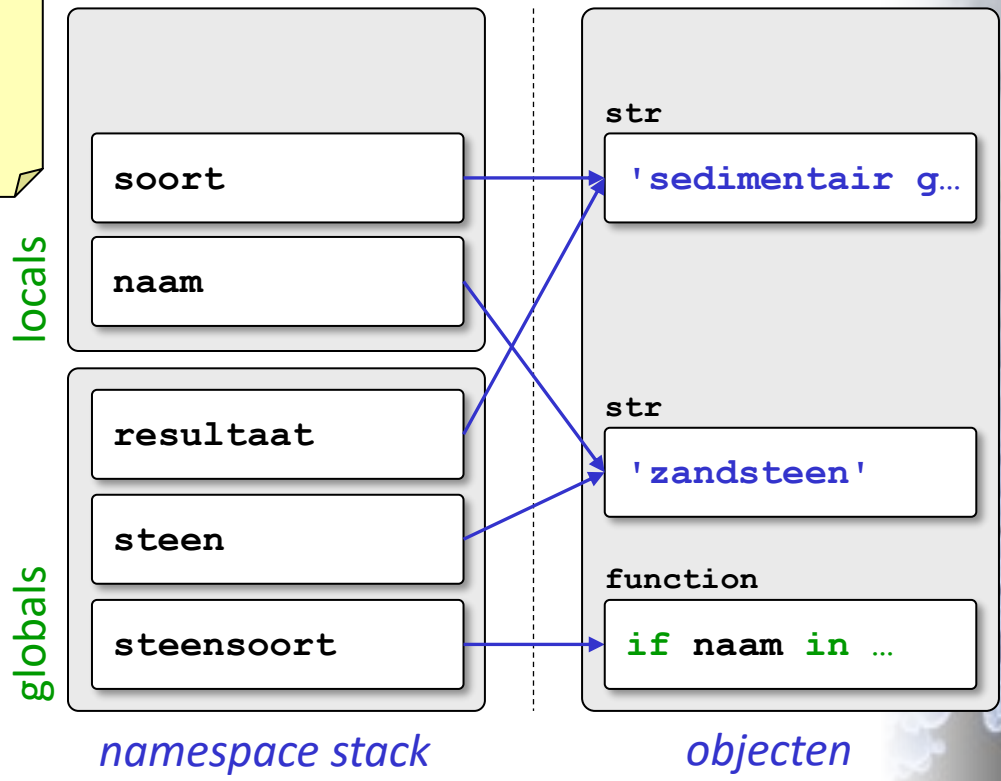
UNIVERSITEIT GENT

Object teruggeven



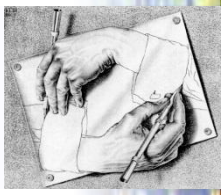
steensoort.py

```
def steensoort(naam):
    if naam in ['basalt', 'graniet']:
        soort = 'stollingsgesteente'
    elif naam in ['zandsteen', 'schalie']:
        soort = 'sedimentair gesteente'
    else:
        soort = 'metamorf gesteente'
    return soort

steen = 'zandsteen'
resultaat = steensoort(steen)
```



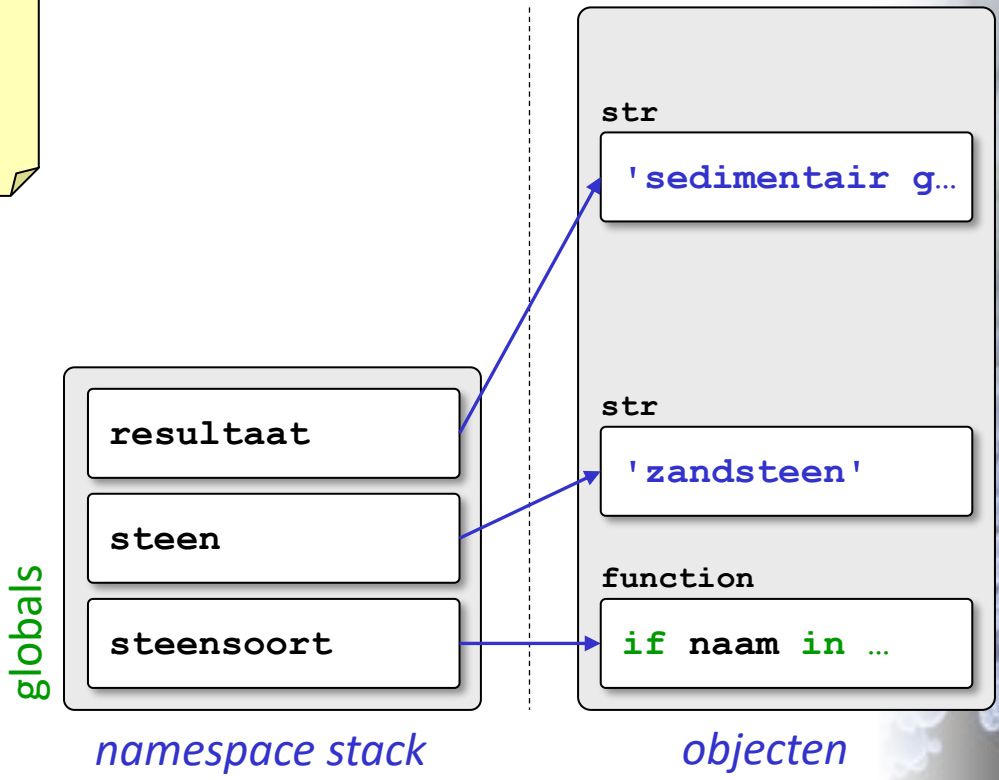
UNIVERSITEIT GENT

Object teruggeven



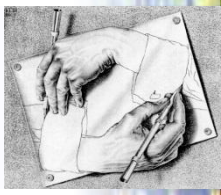
steensoort.py

```
def steensoort(naam):
    if naam in ['basalt', 'graniet']:
        soort = 'stollingsgesteente'
    elif naam in ['zandsteen', 'schalie']:
        soort = 'sedimentair gesteente'
    else:
        soort = 'metamorf gesteente'
    return soort

steen = 'zandsteen'
resultaat = steensoort(steen)
```



UNIVERSITEIT GENT

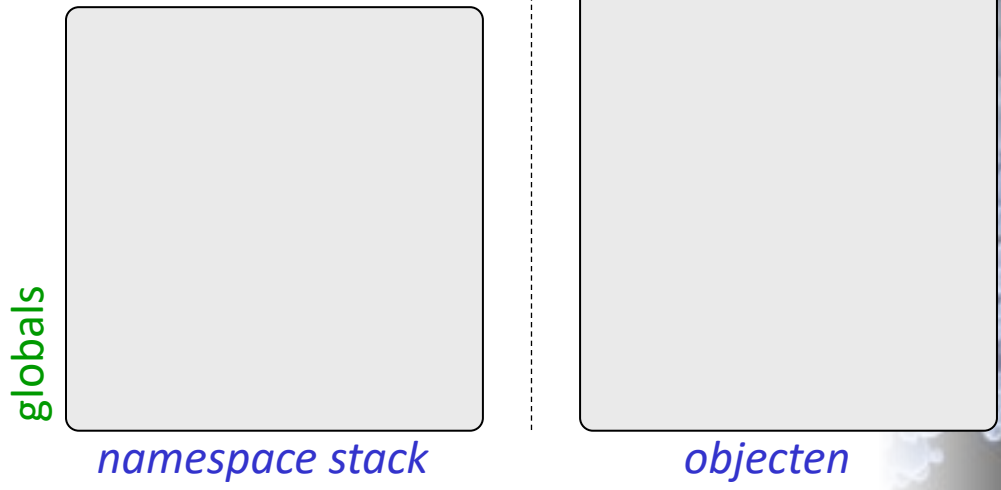
Functies aanroepen



verdubbelen.py

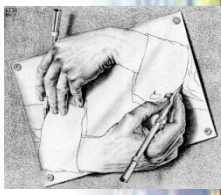
```
def verhoog(a):
    b = a + 1
    return b

def verdubbel(c):
    d = 2 * verhoog(c)
    return d

waarde = 10
resultaat = verdubbel(waarde)
print(resultaat)
```

telkens als we een functie aanroepen wordt er een nieuwe namespace voor de lokale variabelen bovenop de namespace stack geplaatst



UNIVERSITEIT GENT

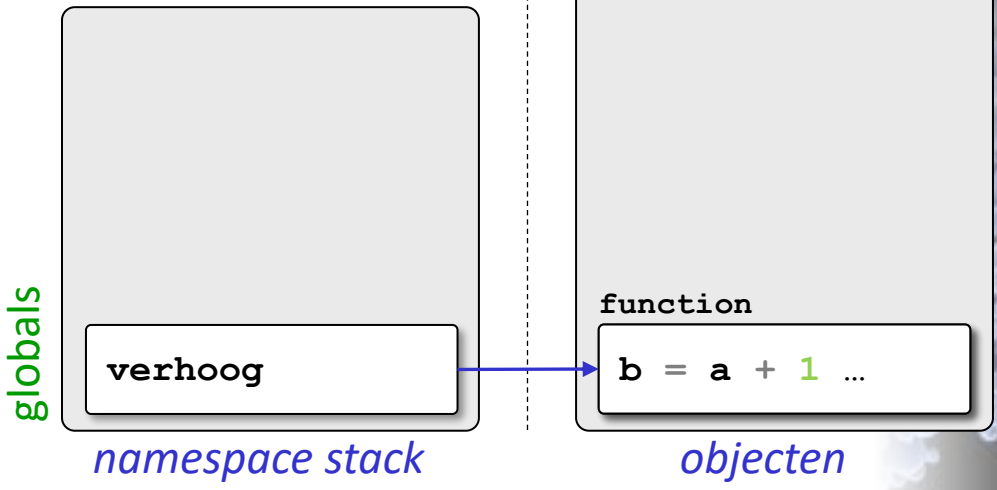
Functies aanroepen



verdubbelen.py

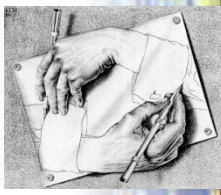
```
def verhoog(a):
    b = a + 1
    return b

def verdubbel(c):
    d = 2 * verhoog(c)
    return d

waarde = 10
resultaat = verdubbel(waarde)
print(resultaat)
```

telkens als we een functie aanroepen wordt er een nieuwe namespace voor de lokale variabelen bovenop de namespace stack geplaatst



UNIVERSITEIT GENT

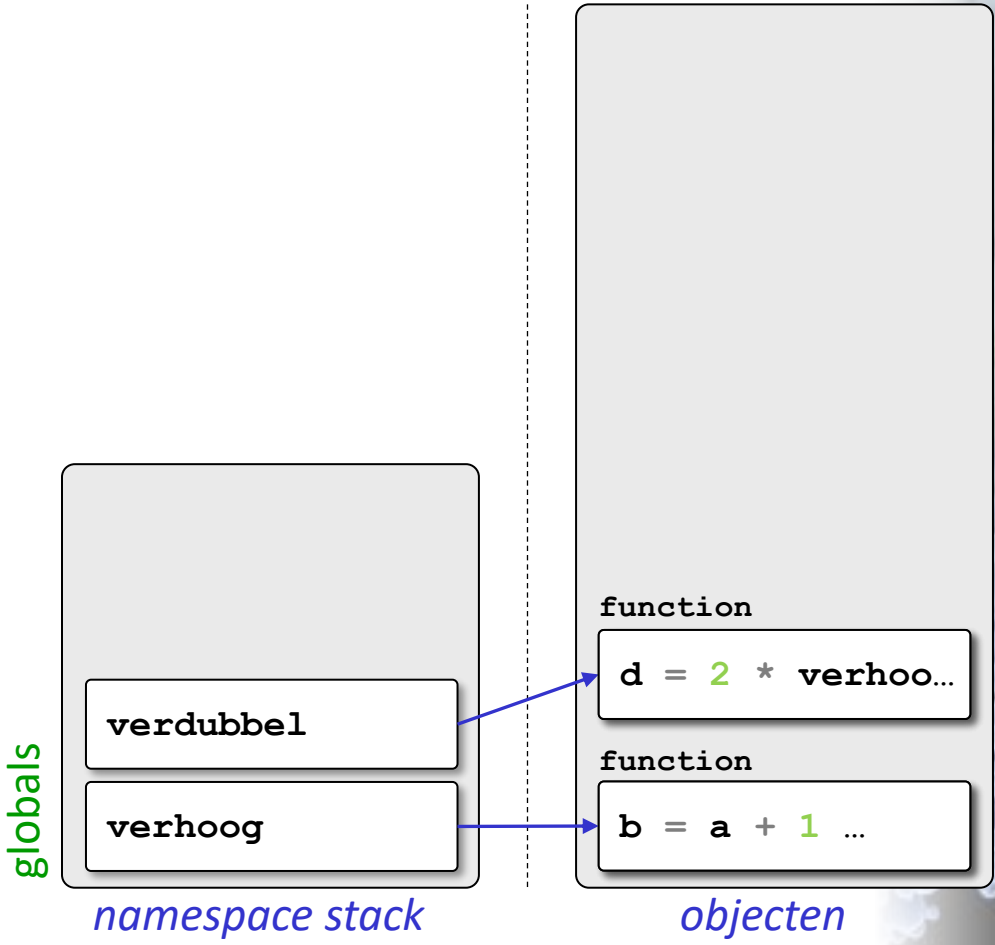
Functies aanroepen



verdubbelen.py

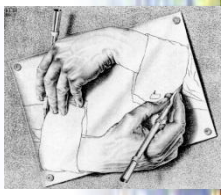
```
def verhoog(a):
    b = a + 1
    return b

def verdubbel(c):
    d = 2 * verhoog(c)
    return d

waarde = 10
resultaat = verdubbel(waarde)
print(resultaat)
```

telkens als we een functie aanroepen wordt er een nieuwe namespace voor de lokale variabelen bovenop de namespace stack geplaatst



UNIVERSITEIT GENT

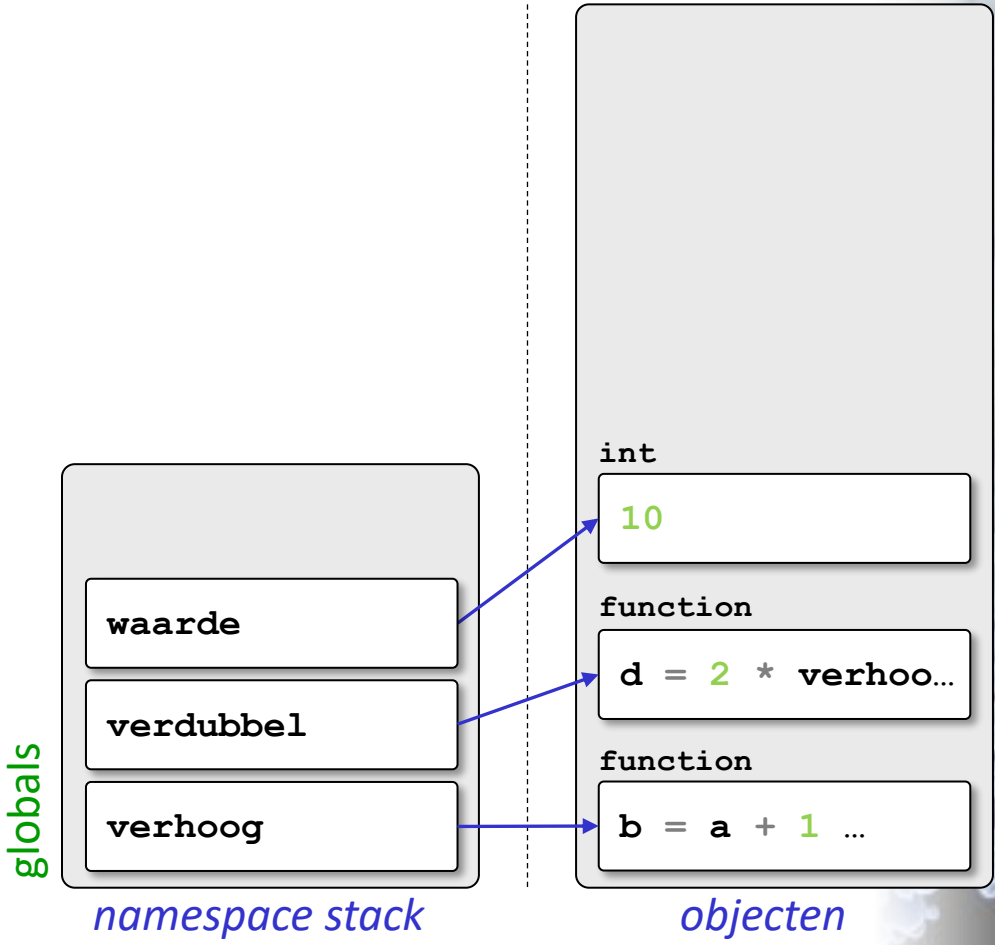
Functies aanroepen



verdubbelen.py

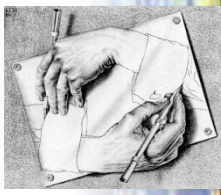
```
def verhoog(a):
    b = a + 1
    return b

def verdubbel(c):
    d = 2 * verhoog(c)
    return d

waarde = 10
resultaat = verdubbel(waarde)
print(resultaat)
```

telkens als we een functie aanroepen wordt er een nieuwe namespace voor de lokale variabelen bovenop de namespace stack geplaatst



UNIVERSITEIT GENT

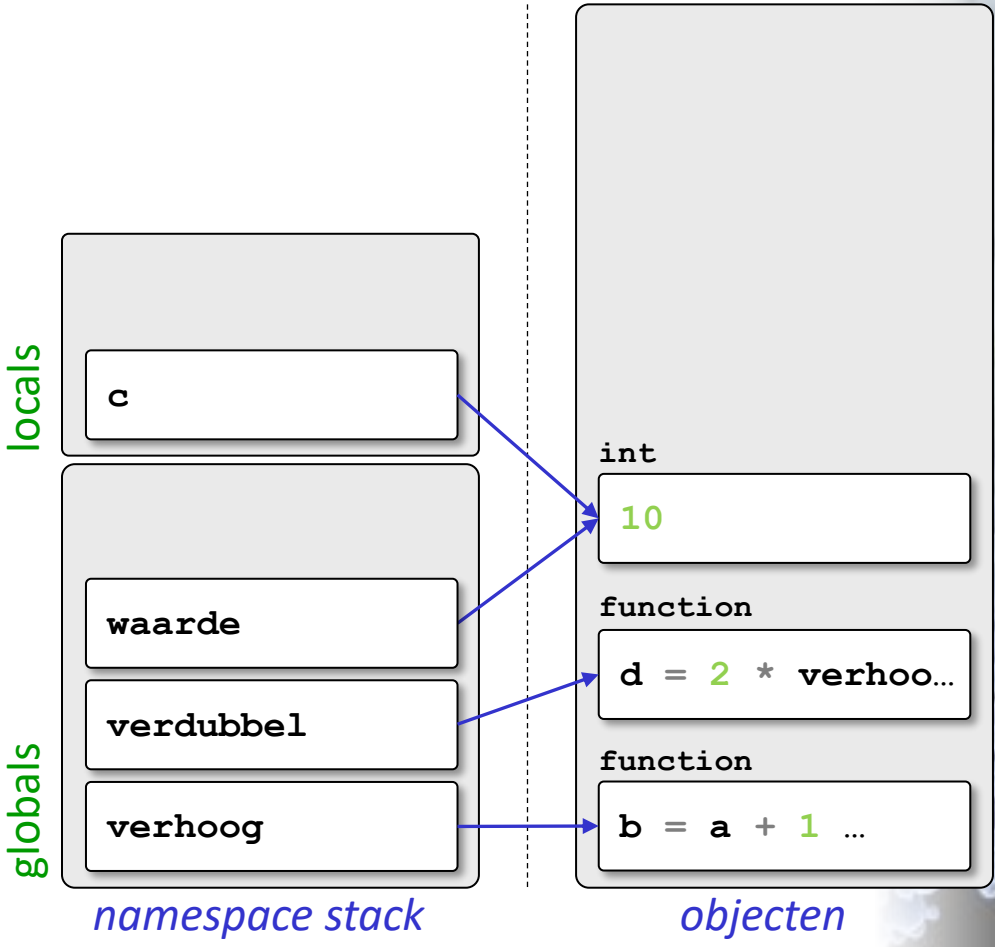
Functies aanroepen



verdubbelen.py

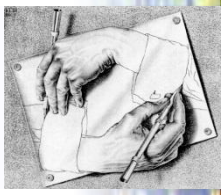
```
def verhoog(a):
    b = a + 1
    return b

def verdubbel(c):
    d = 2 * verhoog(c)
    return d

waarde = 10
resultaat = verdubbel(waarde)
print(resultaat)
```

telkens als we een functie aanroepen wordt er een nieuwe namespace voor de lokale variabelen bovenop de namespace stack geplaatst



UNIVERSITEIT GENT

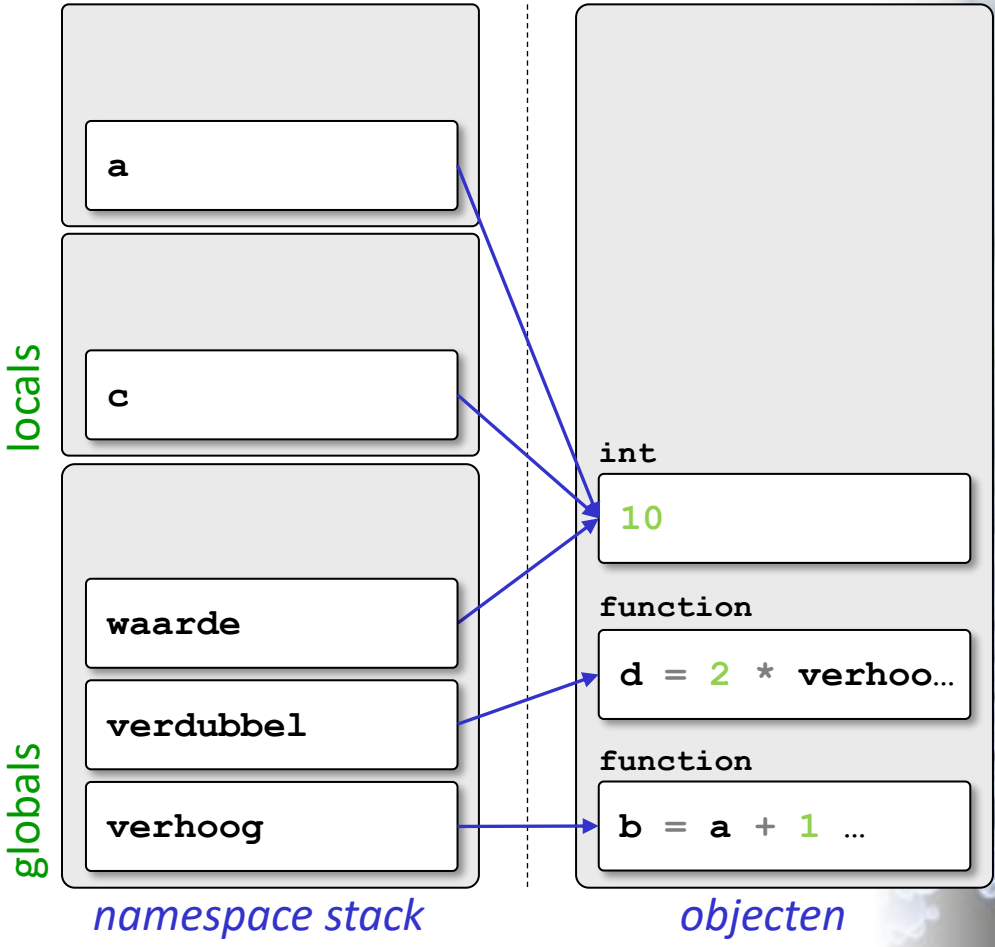
Functies aanroepen



verdubbelen.py

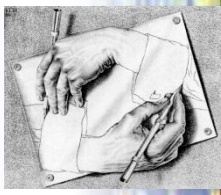
```
def verhoog(a):
    b = a + 1
    return b

def verdubbel(c):
    d = 2 * verhoog(c)
    return d

waarde = 10
resultaat = verdubbel(waarde)
print(resultaat)
```

telkens als we een functie aanroepen wordt er een nieuwe namespace voor de lokale variabelen bovenop de namespace stack geplaatst



UNIVERSITEIT GENT

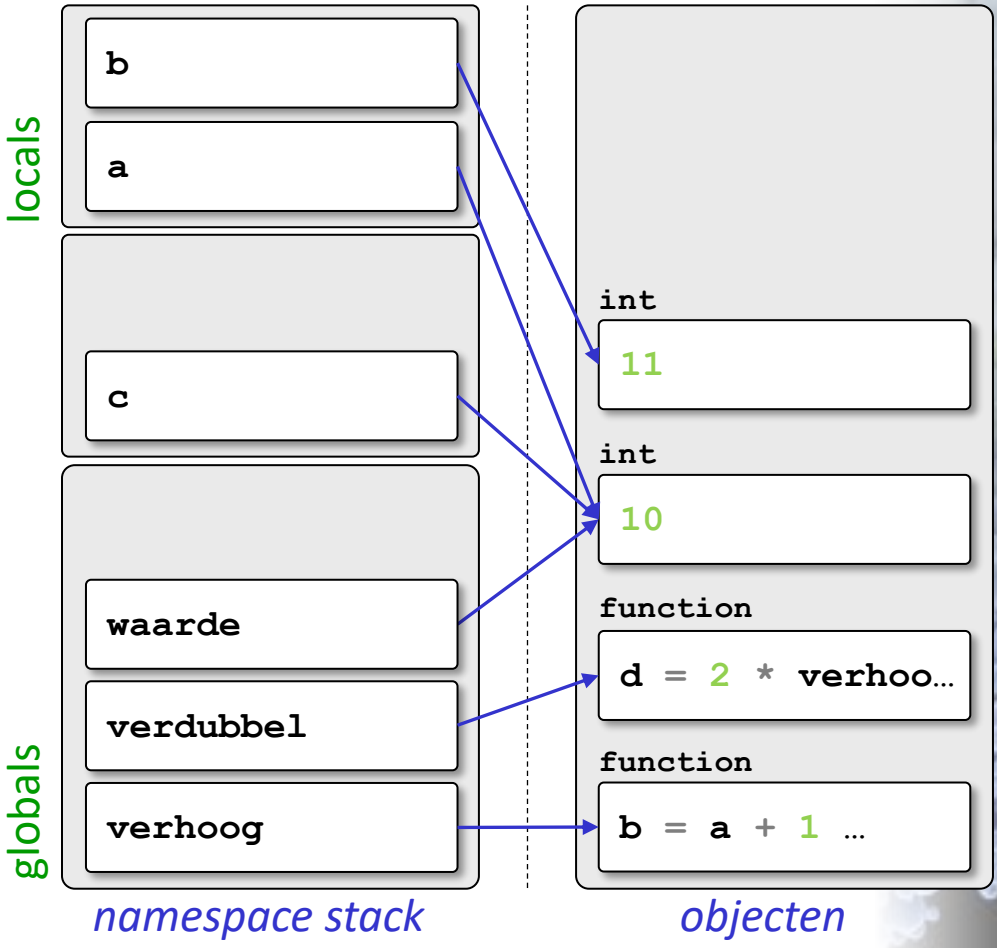
Functies aanroepen



verdubbelen.py

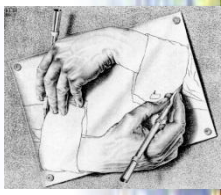
```
def verhoog(a):
    b = a + 1
    return b

def verdubbel(c):
    d = 2 * verhoog(c)
    return d

waarde = 10
resultaat = verdubbel(waarde)
print(resultaat)
```

telkens als we een functie aanroepen wordt er een nieuwe namespace voor de lokale variabelen bovenop de namespace stack geplaatst



UNIVERSITEIT GENT

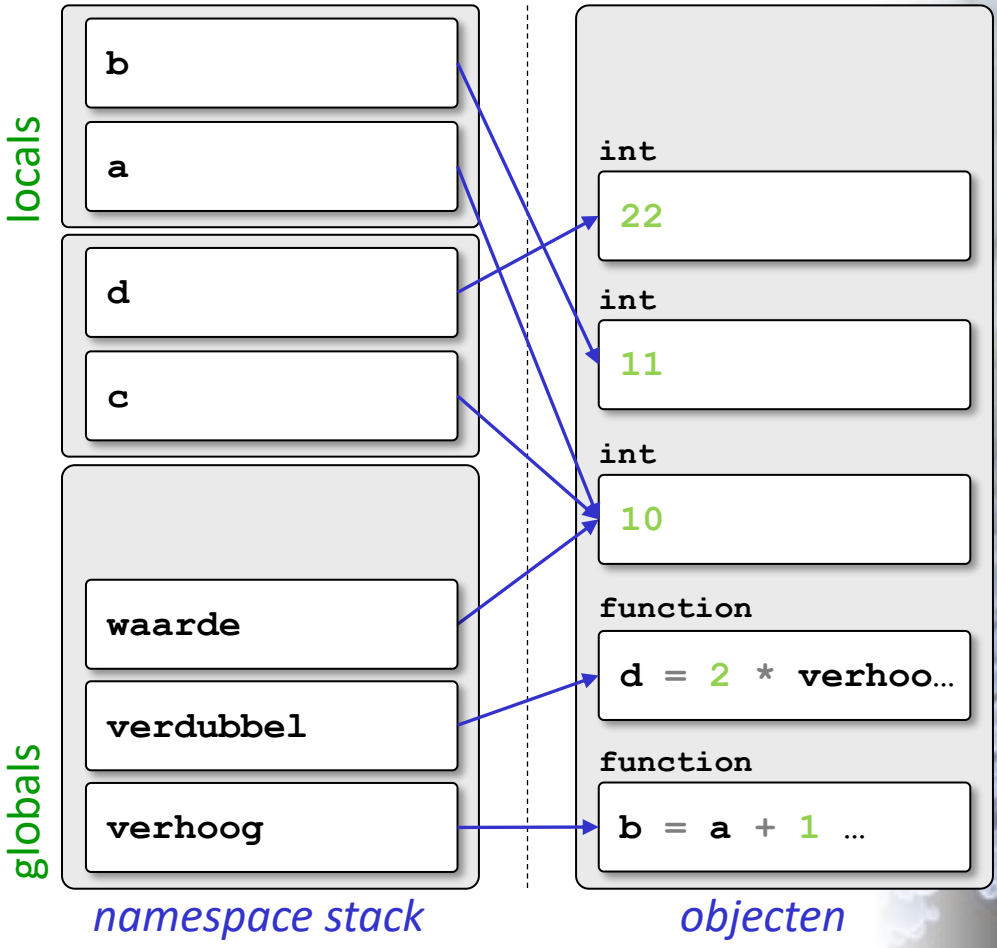
Functies aanroepen



verdubbelen.py

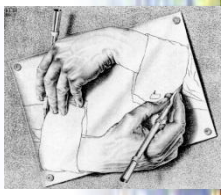
```
def verhoog(a):
    b = a + 1
    return b

def verdubbel(c):
    d = 2 * verhoog(c)
    return d

waarde = 10
resultaat = verdubbel(waarde)
print(resultaat)
```

telkens als we een functie aanroepen wordt er een nieuwe namespace voor de lokale variabelen bovenop de namespace stack geplaatst



UNIVERSITEIT GENT

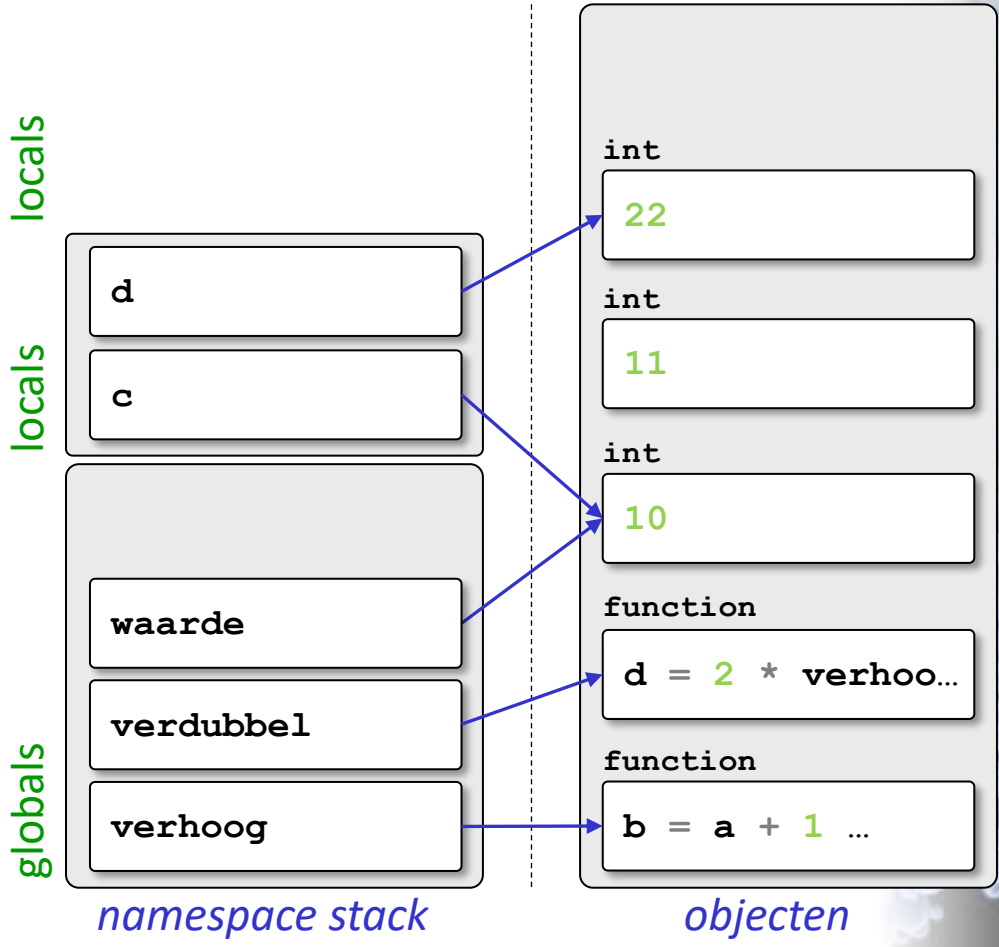
Functies aanroepen



verdubbelen.py

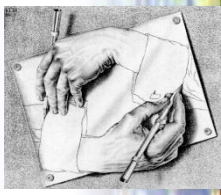
```
def verhoog(a):
    b = a + 1
    return b

def verdubbel(c):
    d = 2 * verhoog(c)
    return d

waarde = 10
resultaat = verdubbel(waarde)
print(resultaat)
```

telkens als we een functie aanroepen wordt er een nieuwe namespace voor de lokale variabelen bovenop de namespace stack geplaatst



UNIVERSITEIT GENT

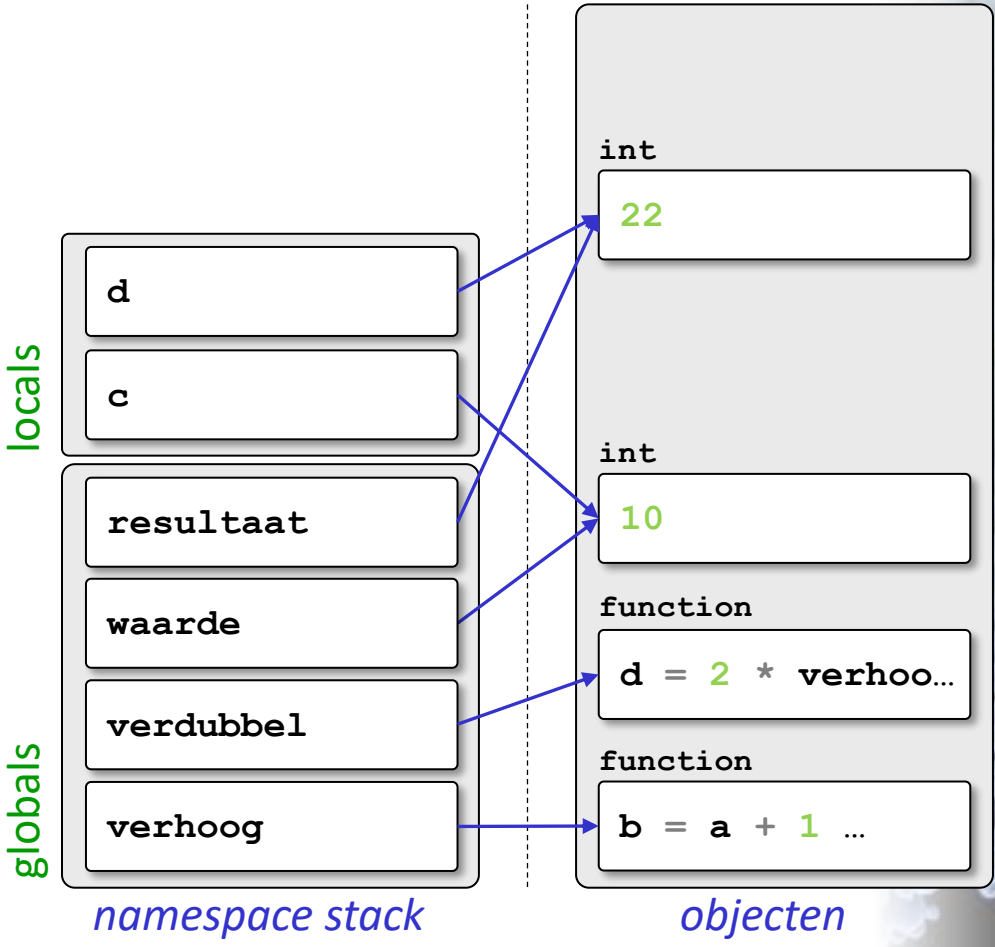
Functies aanroepen



verdubbelen.py

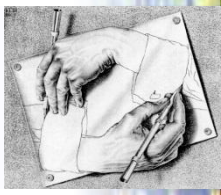
```
def verhoog(a):
    b = a + 1
    return b

def verdubbel(c):
    d = 2 * verhoog(c)
    return d

waarde = 10
resultaat = verdubbel(waarde)
print(resultaat)
```

telkens als we een functie aanroepen wordt er een nieuwe namespace voor de lokale variabelen bovenop de namespace stack geplaatst



UNIVERSITEIT GENT

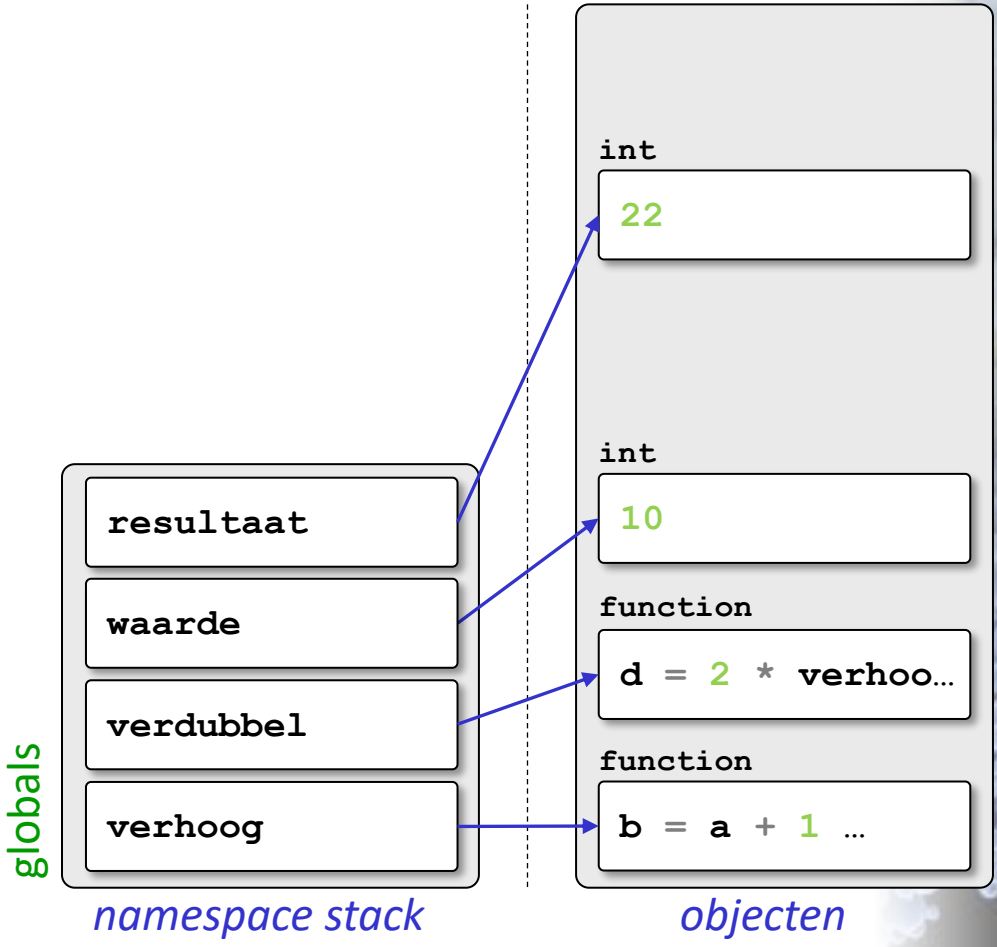
Functies aanroepen



verdubbelen.py

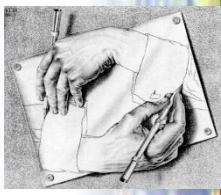
```
def verhoog(a):
    b = a + 1
    return b

def verdubbel(c):
    d = 2 * verhoog(c)
    return d

waarde = 10
resultaat = verdubbel(waarde)
print(resultaat)
```

telkens als we een functie aanroepen wordt er een nieuwe namespace voor de lokale variabelen bovenop de namespace stack geplaatst



UNIVERSITEIT GENT

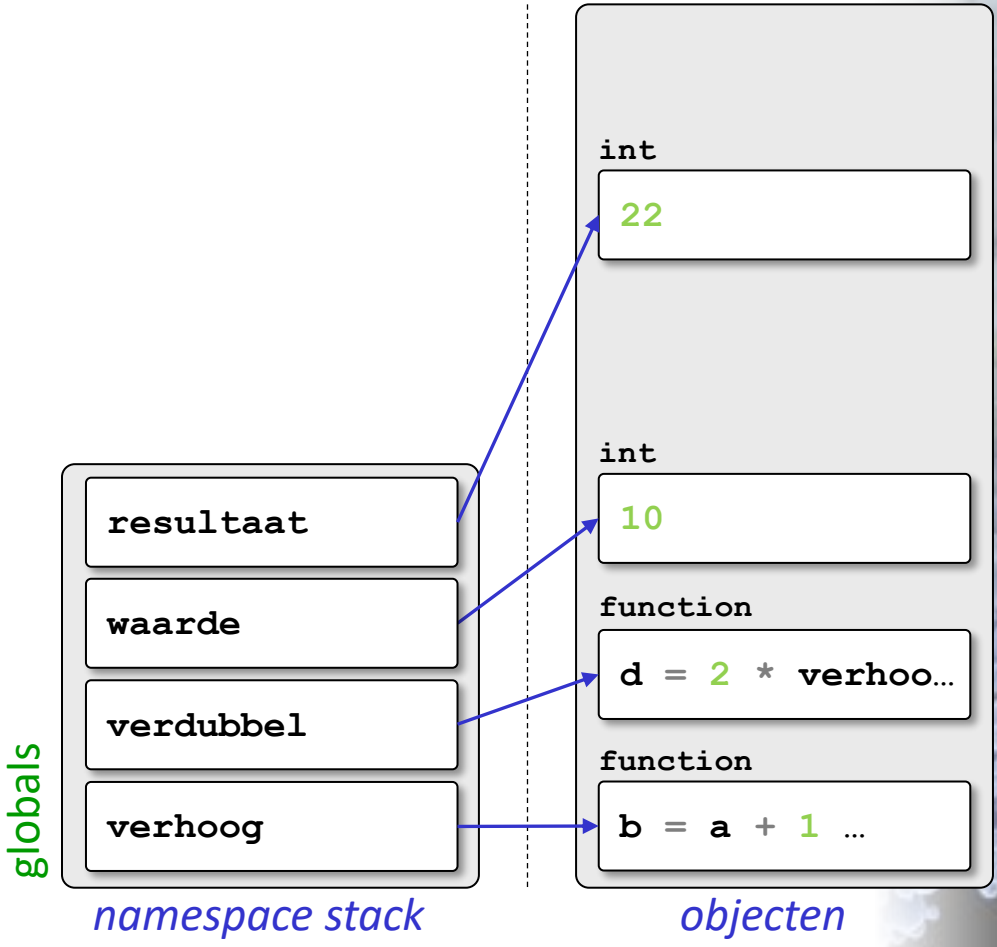
Functies aanroepen



verdubbelen.py

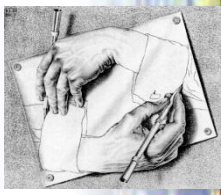
```
def verhoog(a):
    b = a + 1
    return b

def verdubbel(c):
    d = 2 * verhoog(c)
    return d

waarde = 10
resultaat = verdubbel(waarde)
print(resultaat)
```

telkens als we een functie aanroepen wordt er een nieuwe namespace voor de lokale variabelen bovenop de namespace stack geplaatst



UNIVERSITEIT GENT

Scope

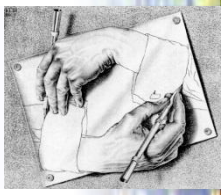


- **scope**: stuk code waarbinnen variabele zichtbaar is
  - wordt bepaald door namespace stack
  - enkel variabelen in bovenste frame (*lokale variabelen*) en onderste frame (*globale variabelen*) zijn zichtbaar

steensoort.py

```
def steensoort(naam):
    if naam in ['basalt', 'graniet']:
        soort = 'stollingsgesteente'
    elif naam in ['zandsteen', 'schalie']:
        soort = 'sedimentair gesteente'
    else:
        soort = 'metamorf gesteente'
    return soort

soort = 'zandsteen'
resultaat = steensoort(soort)
```

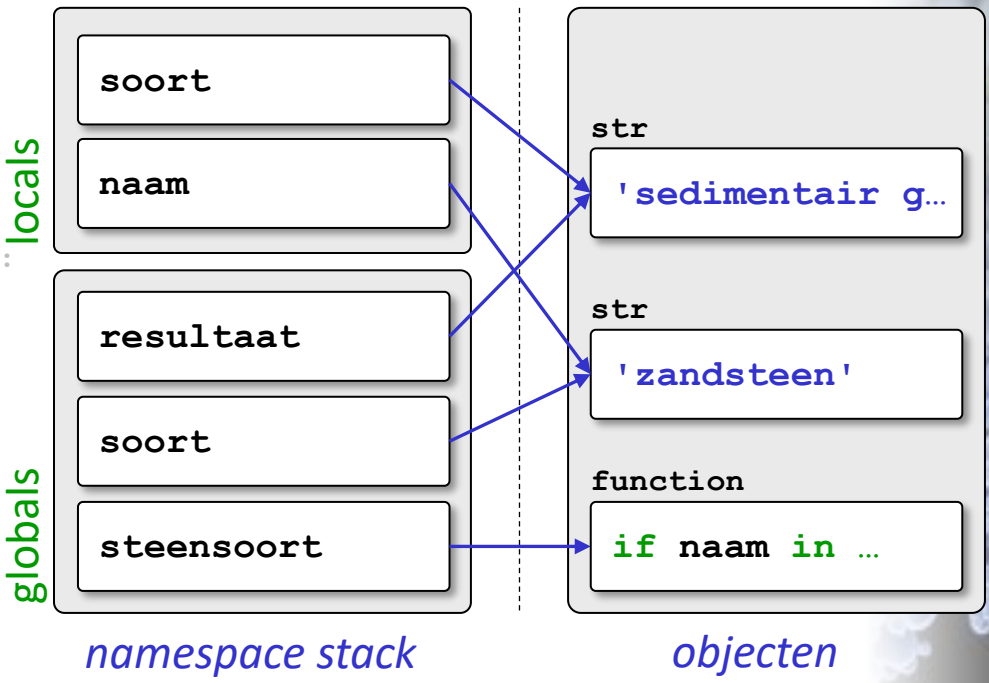
Scope



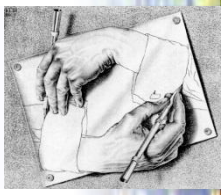
- **scope**: stuk code waarbinnen variabele zichtbaar is
  - wordt bepaald door namespace stack
  - enkel variabelen in bovenste frame (*lokale variabelen*) en onderste frame (*globale variabelen*) zijn zichtbaar
  - lokaal krijgt voorrang boven globaal

steensoort.py

```
def steensoort(naam):
    if naam in ['basalt', 'graniet']:
        soort = 'stollingsgesteente'
    elif naam in ['zandsteen', 'schalie']:
        soort = 'sedimentair gesteente'
    else:
        soort = 'metamorf gesteente'
    return soort

soort = 'zandsteen'
resultaat = steensoort(soort)
```



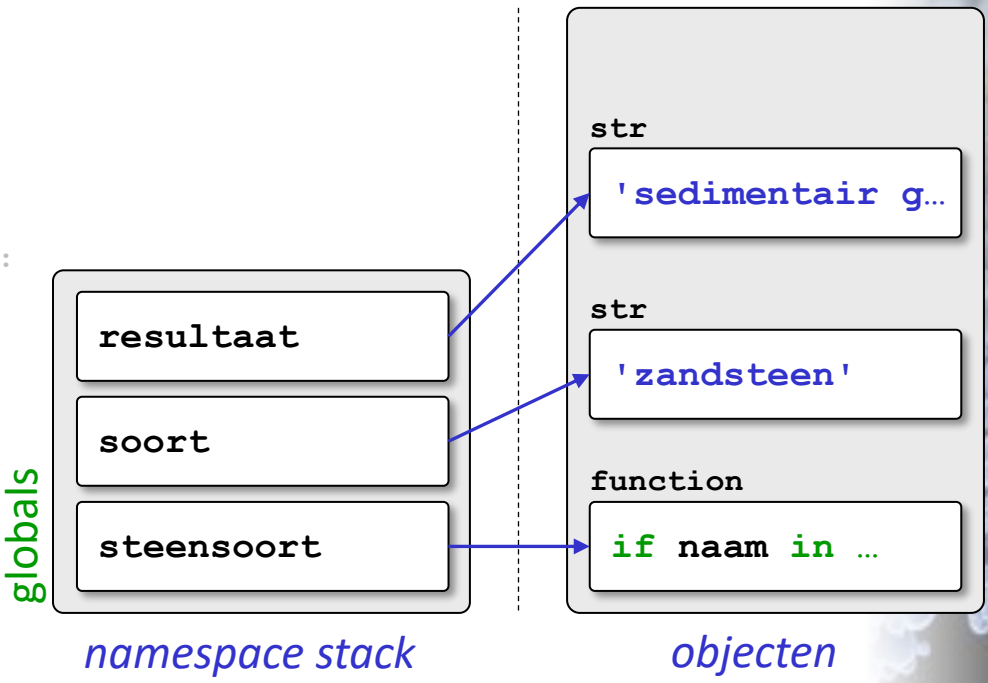
Scope



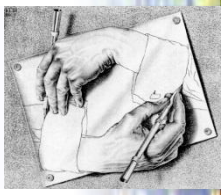
- **scope**: stuk code waarbinnen variabele zichtbaar is
  - wordt bepaald door namespace stack
  - enkel variabelen in bovenste frame (*lokale variabelen*) en onderste frame (*globale variabelen*) zijn zichtbaar
  - lokaal krijgt voorrang boven globaal

steensoort.py

```
def steensoort(naam):
    if naam in ['basalt', 'graniet']:
        soort = 'stollingsgesteente'
    elif naam in ['zandsteen', 'schalie']:
        soort = 'sedimentair gesteente'
    else:
        soort = 'metamorf gesteente'
    return soort

soort = 'zandsteen'
resultaat = steensoort(soort)
```



Expressies door- en teruggeven



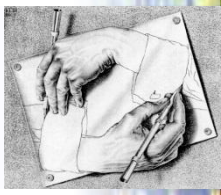
verdubbelen.py

```
def verhoog(a):
    b = a + 1
    return b

def verdubbel(c):
    d = 2 * verhoog(c)
    return d

waarde = 10
resultaat = verdubbel(waarde)
print(resultaat)
```

bij het aanroepen van een functie kunnen expressies als argument doorgegeven worden, en functie kan ook rechtstreeks expressies als resultaat teruggeven

UNIVERSITEIT GENT

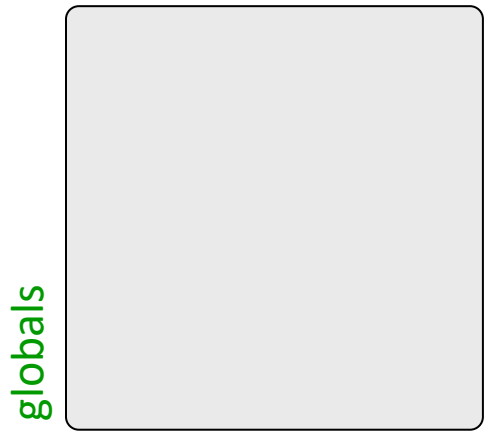
Expressies door- en teruggeven



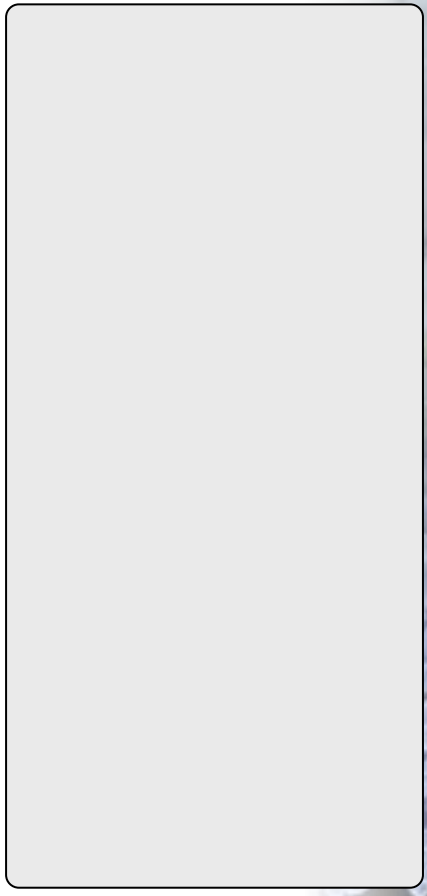
verdubbelen2.py

```
def verhoog(x):
    return x + 1

def verdubbel(x):
    return 2 * verhoog(x)

x = 10
print(verdubbel(x + 3))
```

bij het aanroepen van een functie kunnen expressies als argument doorgegeven worden, en functie kan ook rechtstreeks expressies als resultaat teruggeven



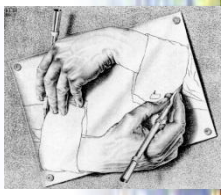
namespace stack



objecten

UNIVERSITEIT GENT

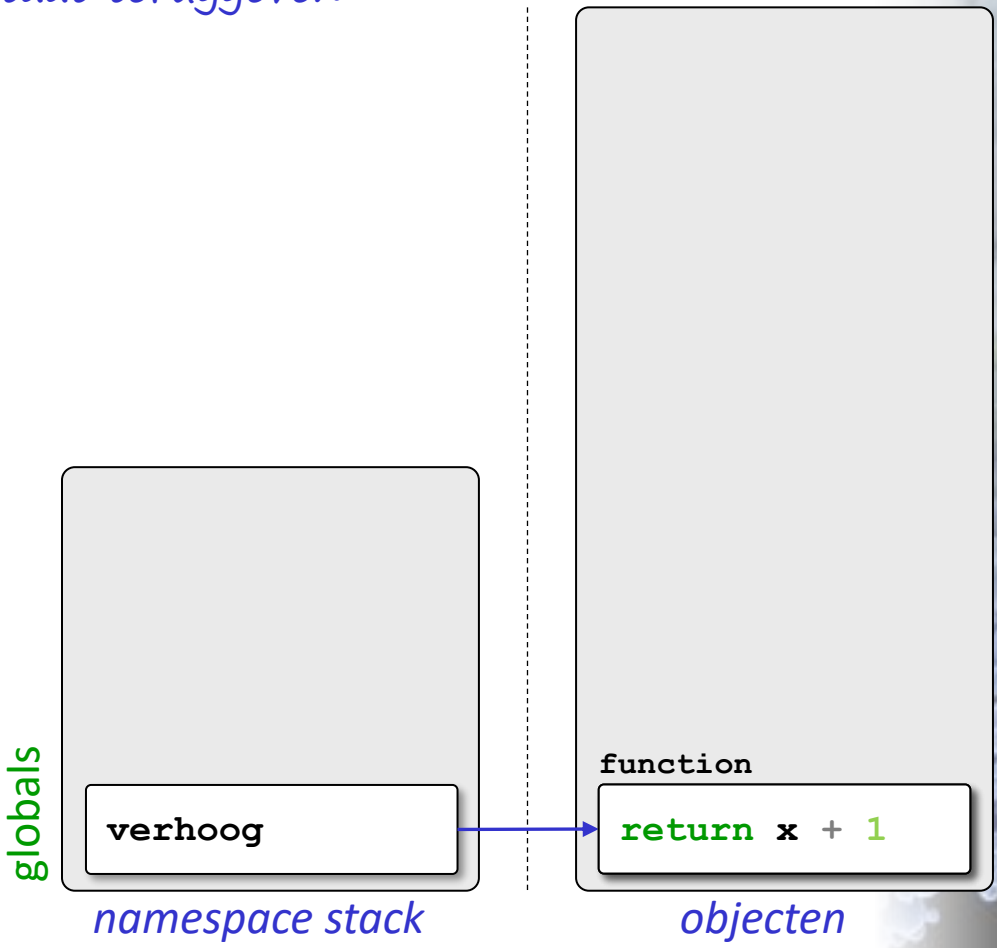
Expressies door- en teruggeven



verdubbelen2.py

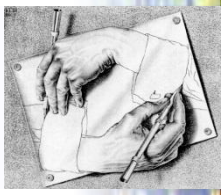
```
def verhoog(x):
    return x + 1

def verdubbel(x):
    return 2 * verhoog(x)

x = 10
print(verdubbel(x + 3))
```

bij het aanroepen van een functie kunnen expressies als argument doorgegeven worden, en functie kan ook rechtstreeks expressies als resultaat teruggeven



UNIVERSITEIT GENT

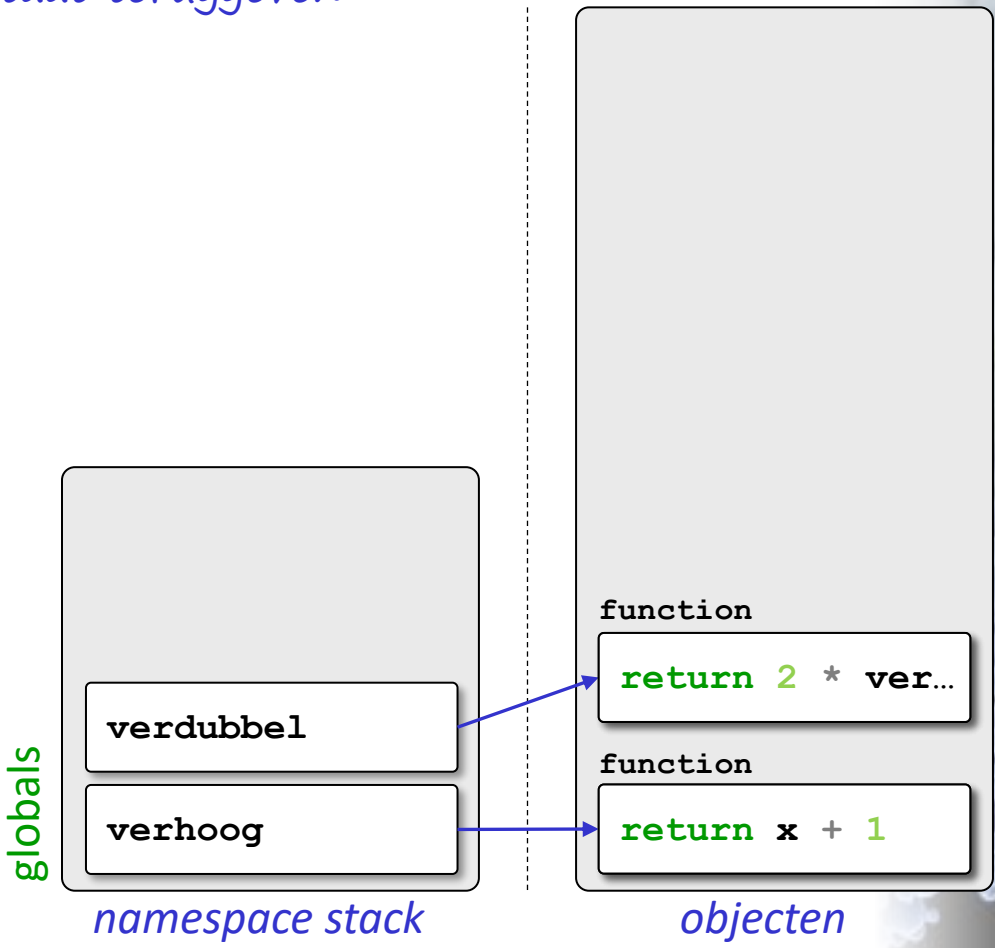
Expressies door- en teruggeven



verdubbelen2.py

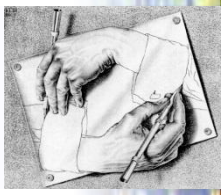
```
def verhoog(x):
    return x + 1

def verdubbel(x):
    return 2 * verhoog(x)

x = 10
print(verdubbel(x + 3))
```

bij het aanroepen van een functie kunnen expressies als argument doorgegeven worden, en functie kan ook rechtstreeks expressies als resultaat teruggeven



UNIVERSITEIT GENT

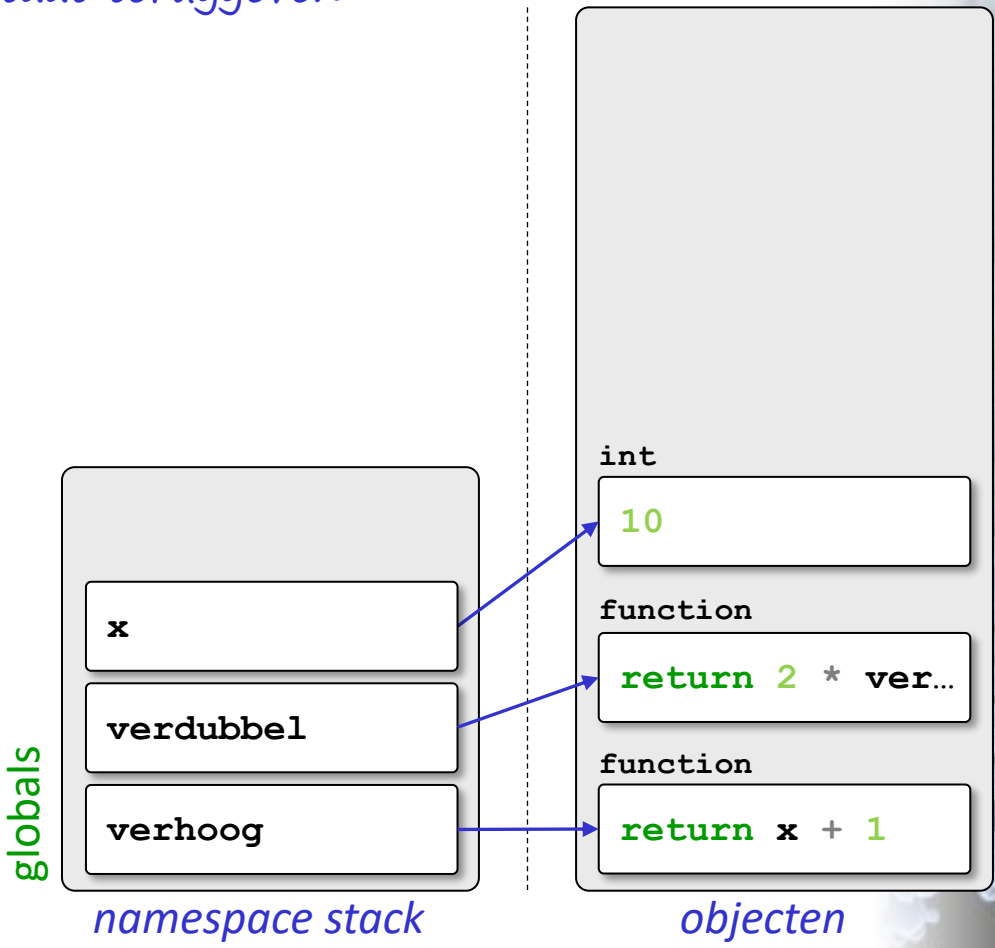
Expressies door- en teruggeven



verdubbelen2.py

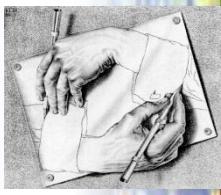
```
def verhoog(x):
    return x + 1

def verdubbel(x):
    return 2 * verhoog(x)

x = 10
print(verdubbel(x + 3))
```

bij het aanroepen van een functie kunnen expressies als argument doorgegeven worden, en functie kan ook rechtstreeks expressies als resultaat teruggeven



UNIVERSITEIT GENT

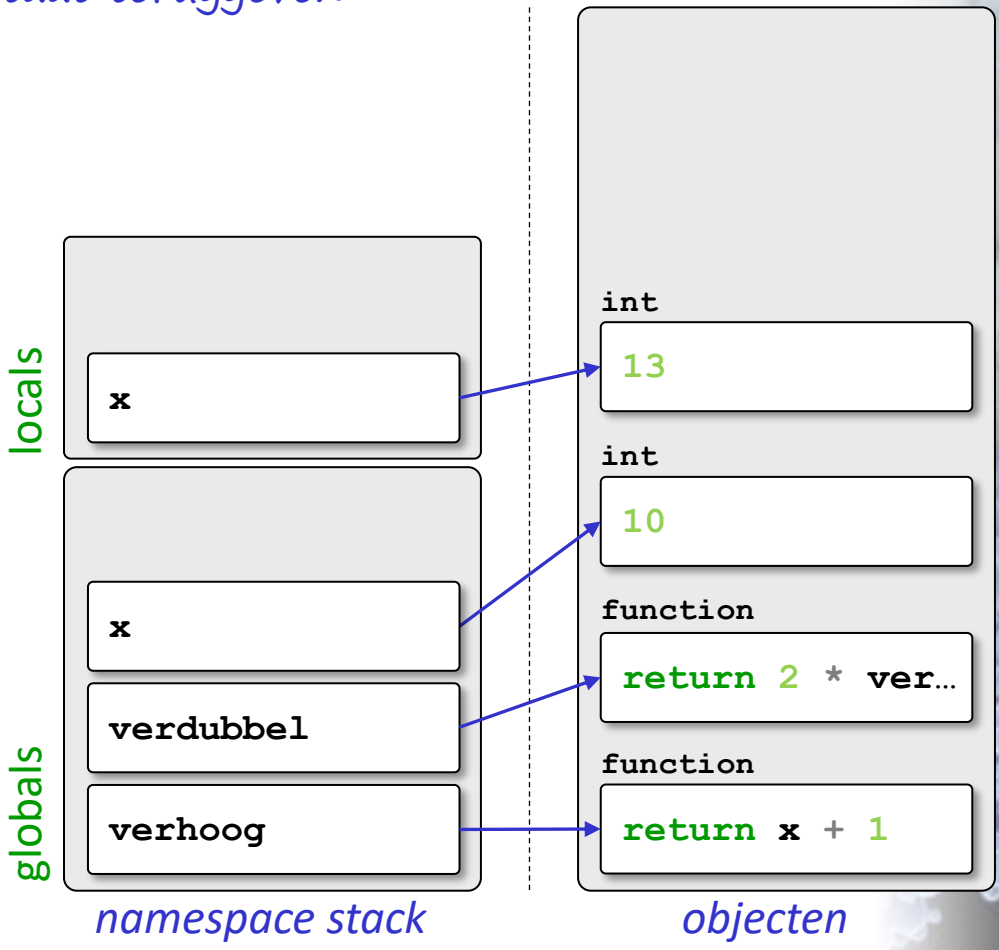
Expressies door- en teruggeven



verdubbelen2.py

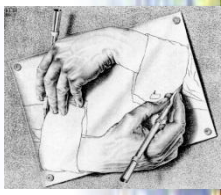
```
def verhoog(x):
    return x + 1

def verdubbel(x):
    return 2 * verhoog(x)

x = 10
print(verdubbel(x + 3))
```

bij het aanroepen van een functie kunnen expressies als argument doorgegeven worden, en functie kan ook rechtstreeks expressies als resultaat teruggeven



UNIVERSITEIT GENT

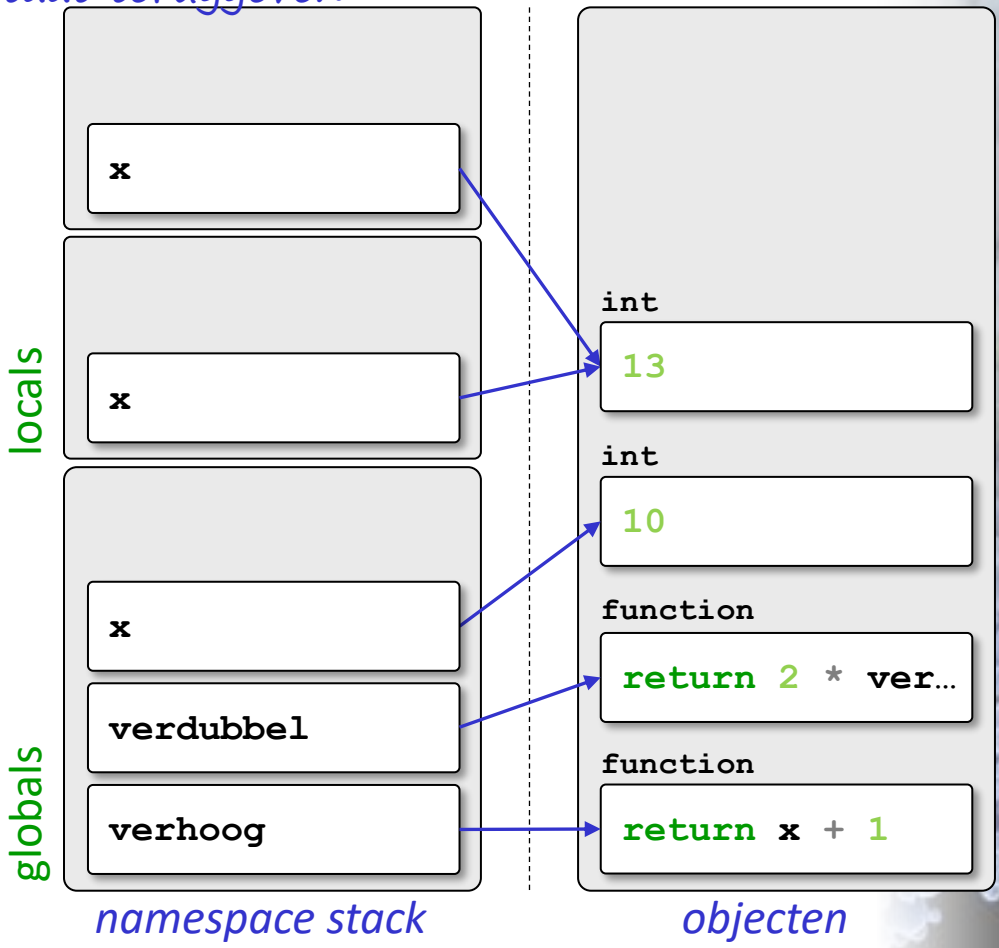
Expressies door- en teruggeven



verdubbelen2.py

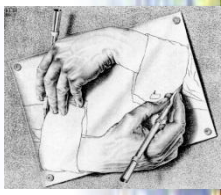
```
def verhoog(x):
    return x + 1

def verdubbel(x):
    return 2 * verhoog(x)

x = 10
print(verdubbel(x + 3))
```

bij het aanroepen van een functie kunnen expressies als argument doorgegeven worden, en functie kan ook rechtstreeks expressies als resultaat teruggeven



UNIVERSITEIT GENT

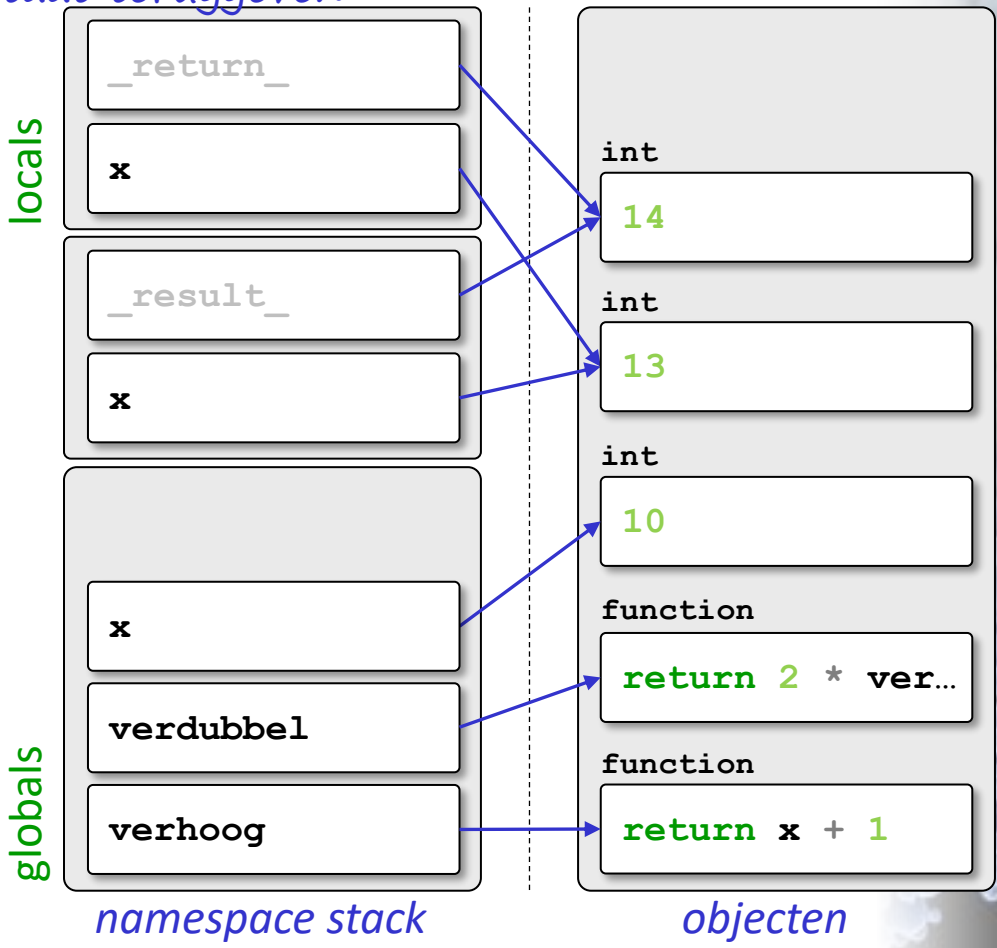
Expressies door- en teruggeven



verdubbelen2.py

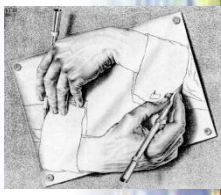
```
def verhoog(x):
    return x + 1

def verdubbel(x):
    return 2 * verhoog(x)

x = 10
print(verdubbel(x + 3))
```

bij het aanroepen van een functie kunnen expressies als argument doorgegeven worden, en functie kan ook rechtstreeks expressies als resultaat teruggeven



UNIVERSITEIT GENT

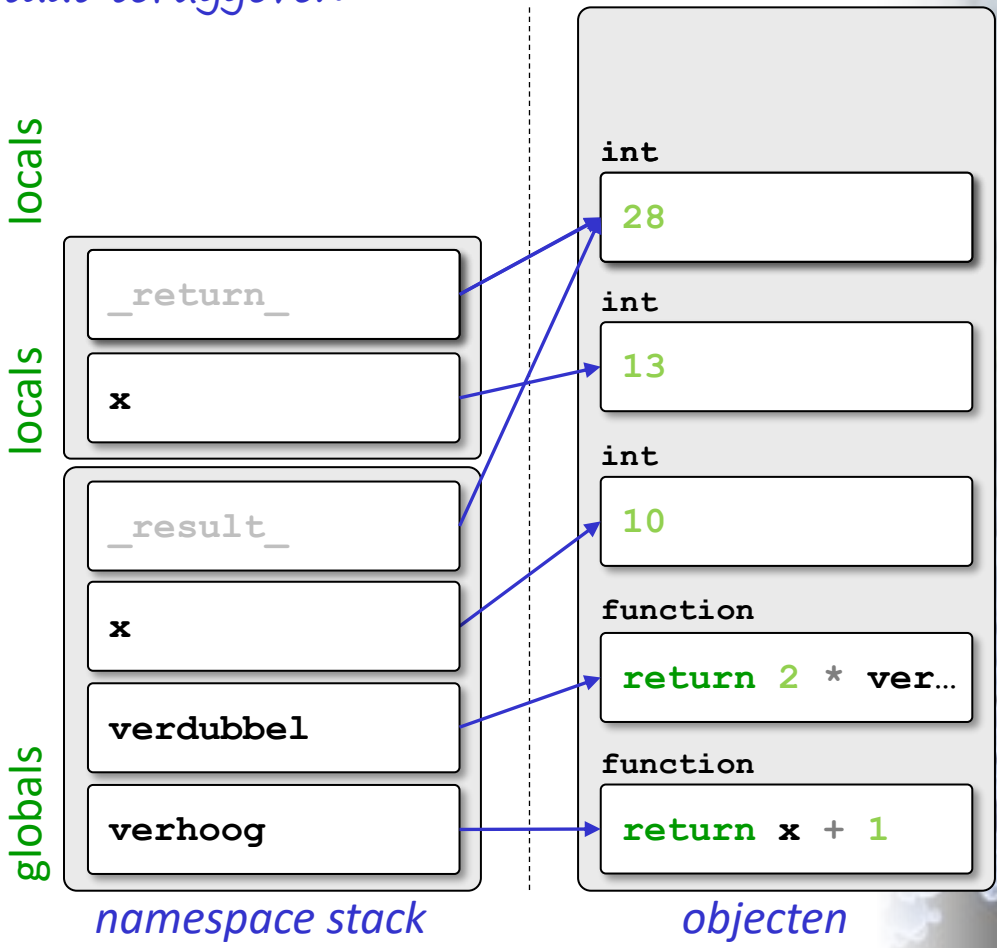
Expressies door- en teruggeven



verdubbelen2.py

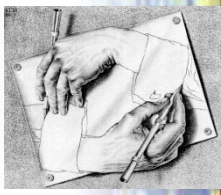
```
def verhoog(x):
    return x + 1

def verdubbel(x):
    return 2 * verhoog(x)

x = 10
print(verdubbel(x + 3))
```

bij het aanroepen van een functie kunnen expressies als argument doorgegeven worden, en functie kan ook rechtstreeks expressies als resultaat teruggeven



UNIVERSITEIT GENT

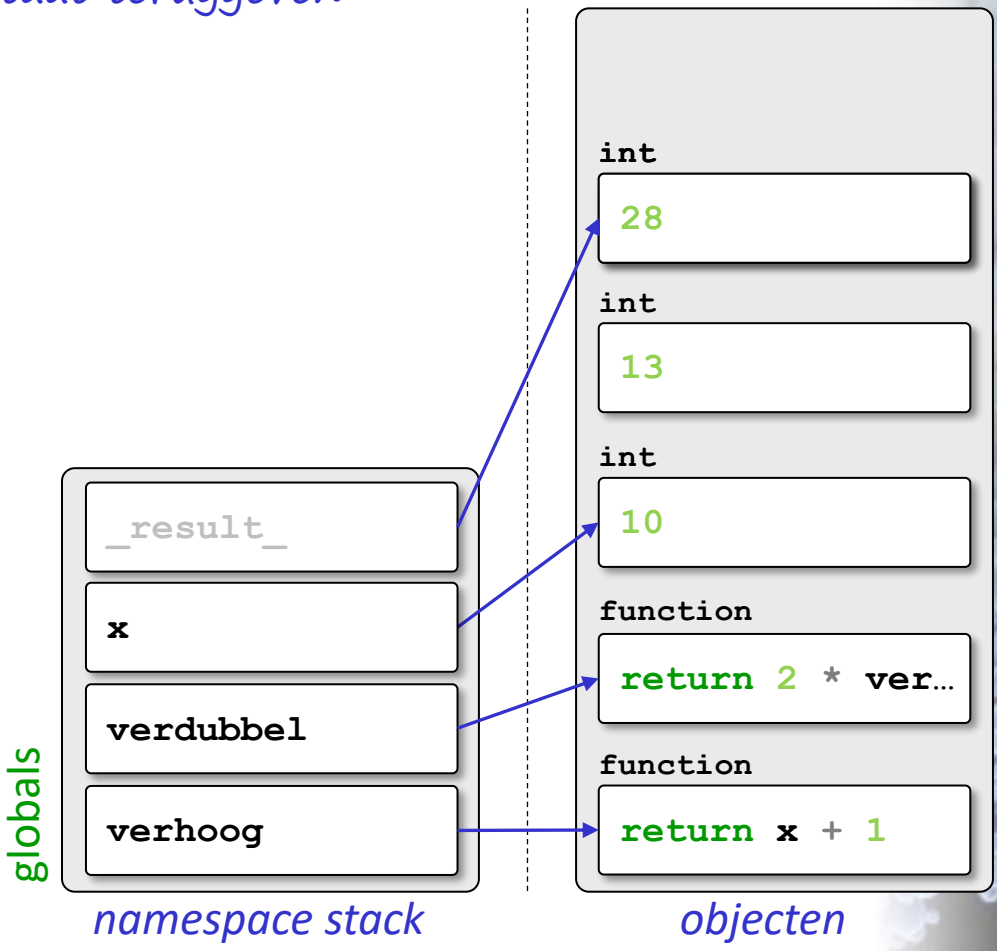
Expressies door- en teruggeven



verdubbelen2.py

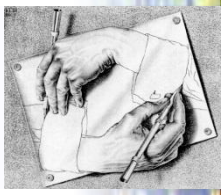
```
def verhoog(x):
    return x + 1

def verdubbel(x):
    return 2 * verhoog(x)

x = 10
print(verdubbel(x + 3))
```

bij het aanroepen van een functie kunnen expressies als argument doorgegeven worden, en functie kan ook rechtstreeks expressies als resultaat teruggeven



UNIVERSITEIT GENT

Object teruggeven

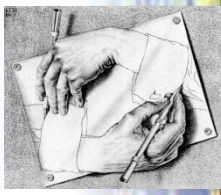


- return statement
  - evalueert expressie en geeft waarde als resultaat terug
  - daarna wordt lokale scope van de functie onmiddellijk verwijderd van namespace stack
- kan overal binnen body van functie voorkomen
- kan meerdere keren binnen body van functie voorkomen

teken.py

```
def teken(getal):
    if getal > 0:
        return 1
    elif getal == 0:
        return 0
    else:
        return -1

print(teken(3))
```

Object teruggeven

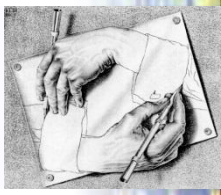


- return statement
  - evalueert expressie en geeft waarde als resultaat terug
  - daarna wordt lokale scope van de functie onmiddellijk verwijderd van namespace stack
- kan overal binnen body van functie voorkomen
- kan meerdere keren binnen body van functie voorkomen

teken.py

```
def teken(getal):
    if getal > 0:
        return 1
    elif getal == 0:
        return 0
    else:
        return -1

print(teken(-9))
```

Object teruggeven

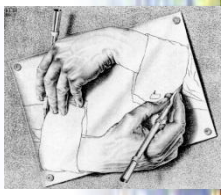


- overmatig gebruik maakt functie onleesbaar
- geen algemene richtlijnen, maar …
  - klein aantal in begin om speciale gevallen op te vangen
  - één op einde om algemeen resultaat terug te geven

teken.py

```
def teken(getal):
    if getal > 0:
        return 1
    elif getal == 0:
        return 0
    else:
        return -1

print(teken(-9))
```

Object teruggeven



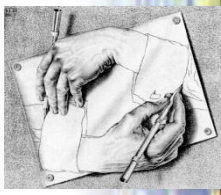
- functies geven altijd een resultaat terug

teken.py

```
def teken(getal):
    if getal > 0:
        return 1
    elif getal == 0:
        return 0
#    else:
#        return -1

print(teken(3))
```

UNIVERSITEIT GENT

Object teruggeven



- functies geven altijd een resultaat terug
  - indien functie niet expliciet waarde teruggeeft, dan geeft Python de waarde **`None`** terug

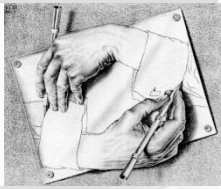
teken.py

```
def teken(getal):
    if getal > 0:
        return 1
    elif getal == 0:
        return 0
#    else:
#        return -1

print(teken(-9))
```

None

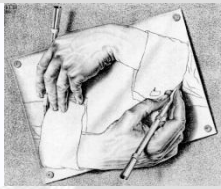
UNIVERSITEIT GENT

Meirp

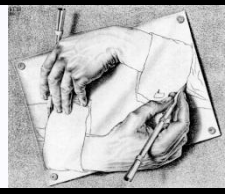


13, 17, 31, 37, 71, 73, 79, 97, 107, 113, 149, 157, 167, 179, 199, 311, 337, 347, 359, 389, 701, 709, 733, 739, 743, 751, 761, 769, 907, 937, 941, 953, 967, 971, 983, 991, 1009, 1021, 1031, 1033, 1061, 1069, 1091, 1097, 1103, 1109, 1151, 1153, 1181, 1193, …

GSMoniemen



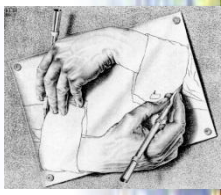
Slofpeut



RDIAENG

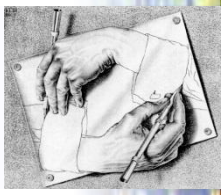
Aoccdrnig to a rscheearch at an Elingsh uinervtisy, it deosn't mttaer in waht oredr the ltteers in a wrod are. The olny iprmoetnt tihng is taht the frist and lsat ltteer is at the rghit pclae. The rset can be a toatl mses and you can sitll raed it wouthit porbelm. Tihs is bcuseae we do not raed ervey lteter by it slef but the wrod as a wlohe. ceehiro.

Huiswerk (volgende les)



- lees handboek
  - hoofdstuk 7 (lijsten)
- lees omschrijving van voorbeeldopgaven (reeks 6)
  - Chocoladerepen
  - Circadiale permutator

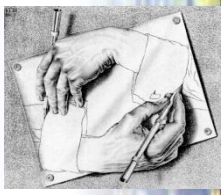
Huiswerk (werkcolleges)



- videohandleidingen
  - test-driven development
  - debuggen van broncode (deel 2)
- opgelegde oefeningen reeks 5 (functies)
  - ISBN (demo)
  - Woordsommen
  - De laatste banaan
  - Achteruitkijkspiegel
  - Inventaris
  - Blinkenlights

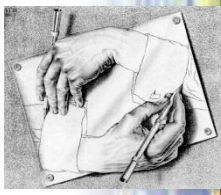
UNIVI
GENT

Vragen of opmerkingen?



UNIVERSITEIT GENT

The sky is the limit …



"He who wonders, discovers
that this in itself is a wonder."

— MC Escher

UNIVERSITEIT
GENT