Universiteit Gent
Faculteit Wetenschappen

Programmeren in Python

express yourself

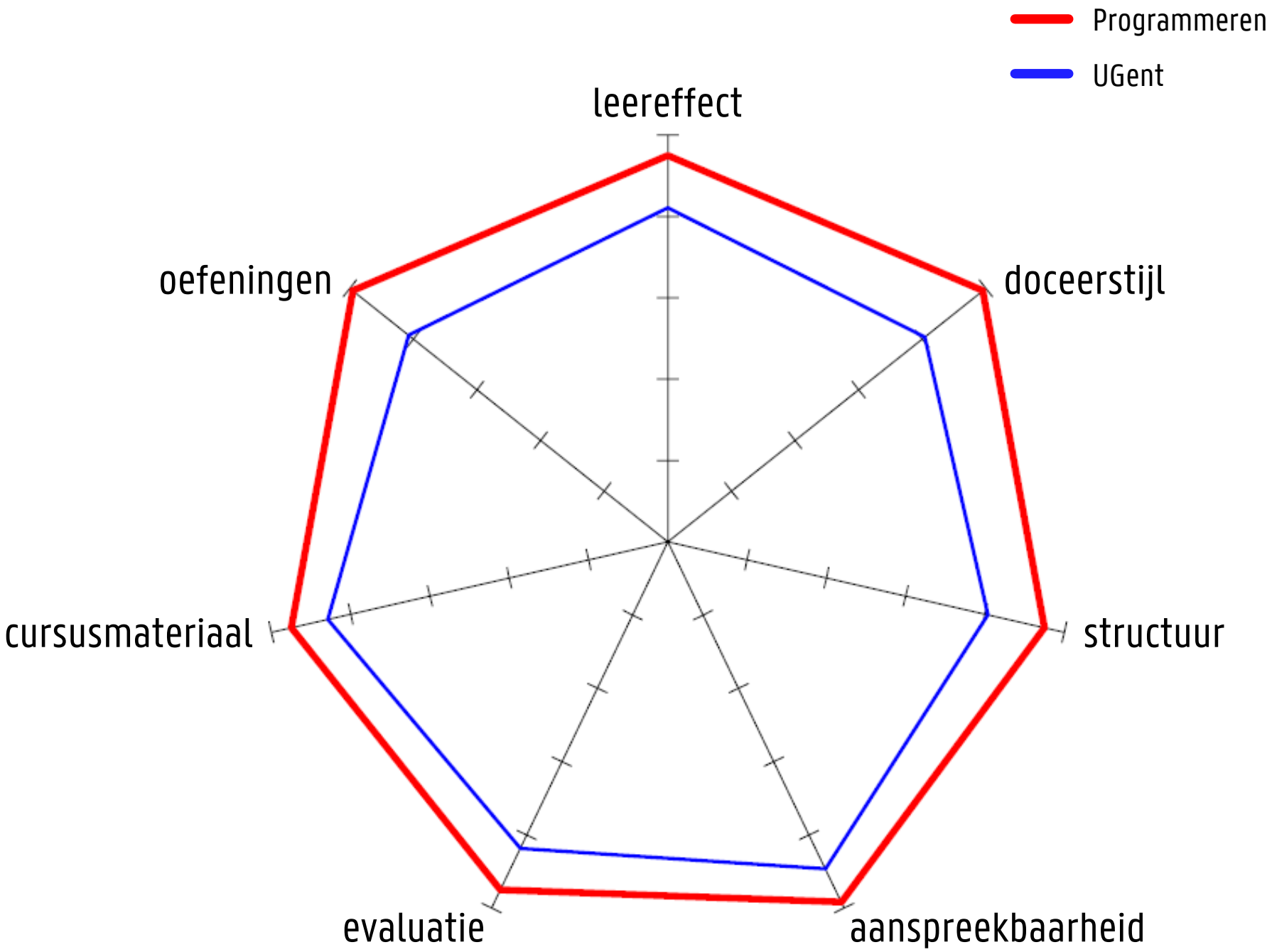
prof. dr. Peter Dawyndt

peter.dawyndt@ugent.be

@dawyndt

UNIVERSITEIT GENT


Ooooh! The last bug!


Programmeren
UGent
leereffect
doceerstijl
structuur
aanspreekbaarheid
evaluatie
cursusmateriaal
oefeningen

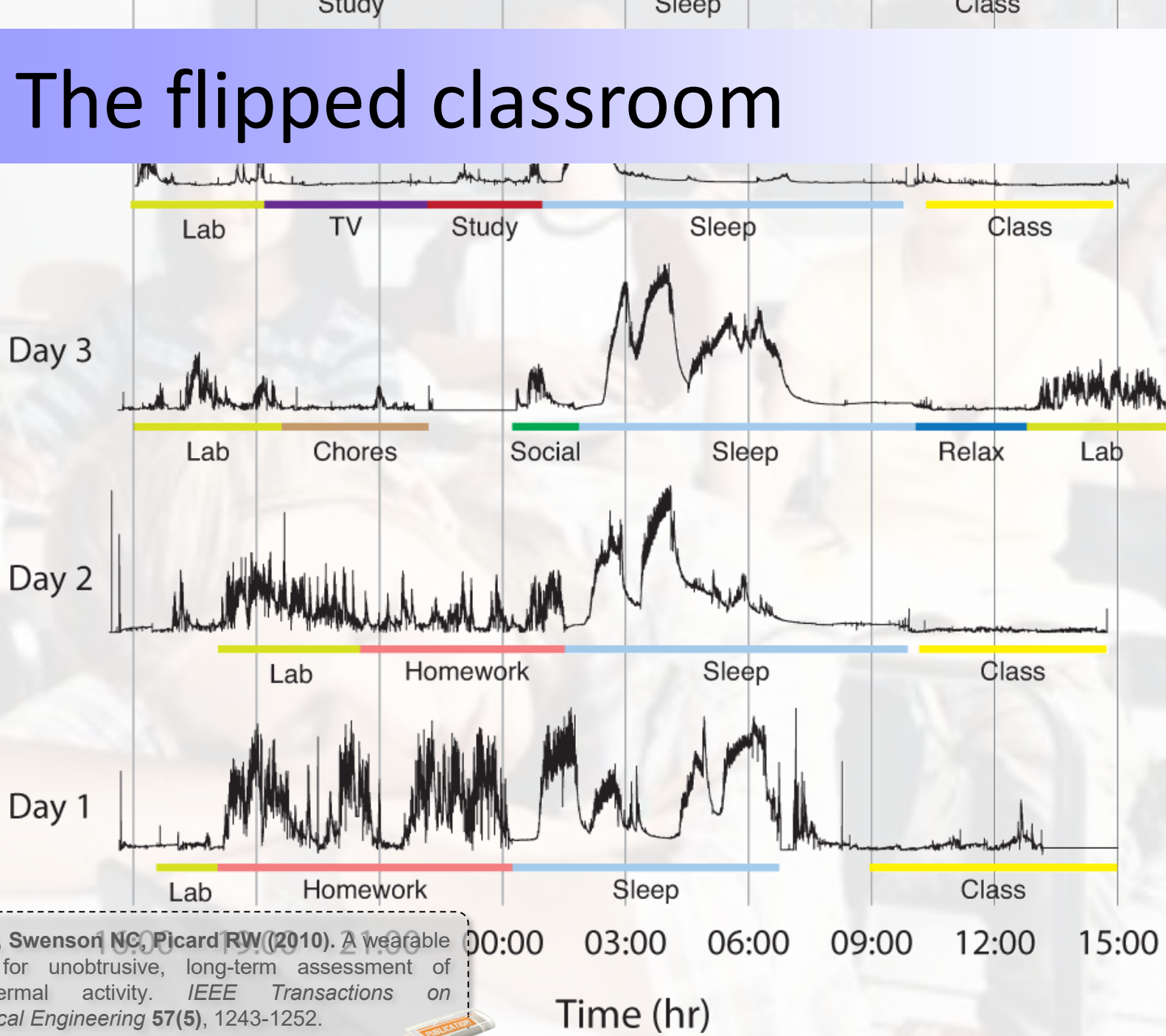
The flipped classroom

traditional classroom (ideal world)

lecture | study | lecture | study | lecture | study | study | exam

UNIVERSITEIT GENT

The flipped classroom



Poh MZ, Swenson NC, Picard RW (2010). A wearable sensor for unobtrusive, long-term assessment of electrodermal activity. *IEEE Transactions on Biomedical Engineering* **57(5)**, 1243-1252.

The flipped classroom

traditional classroom (ideal world)

lecture | study | lecture | study | lecture | study | study | exam

UNIVERSITEIT GENT

The flipped classroom

traditional classroom (real world)

lecture

lecture

lecture

study

exam

flipped classroom

practice

study

lecture

practice

study

lecture

practice

study

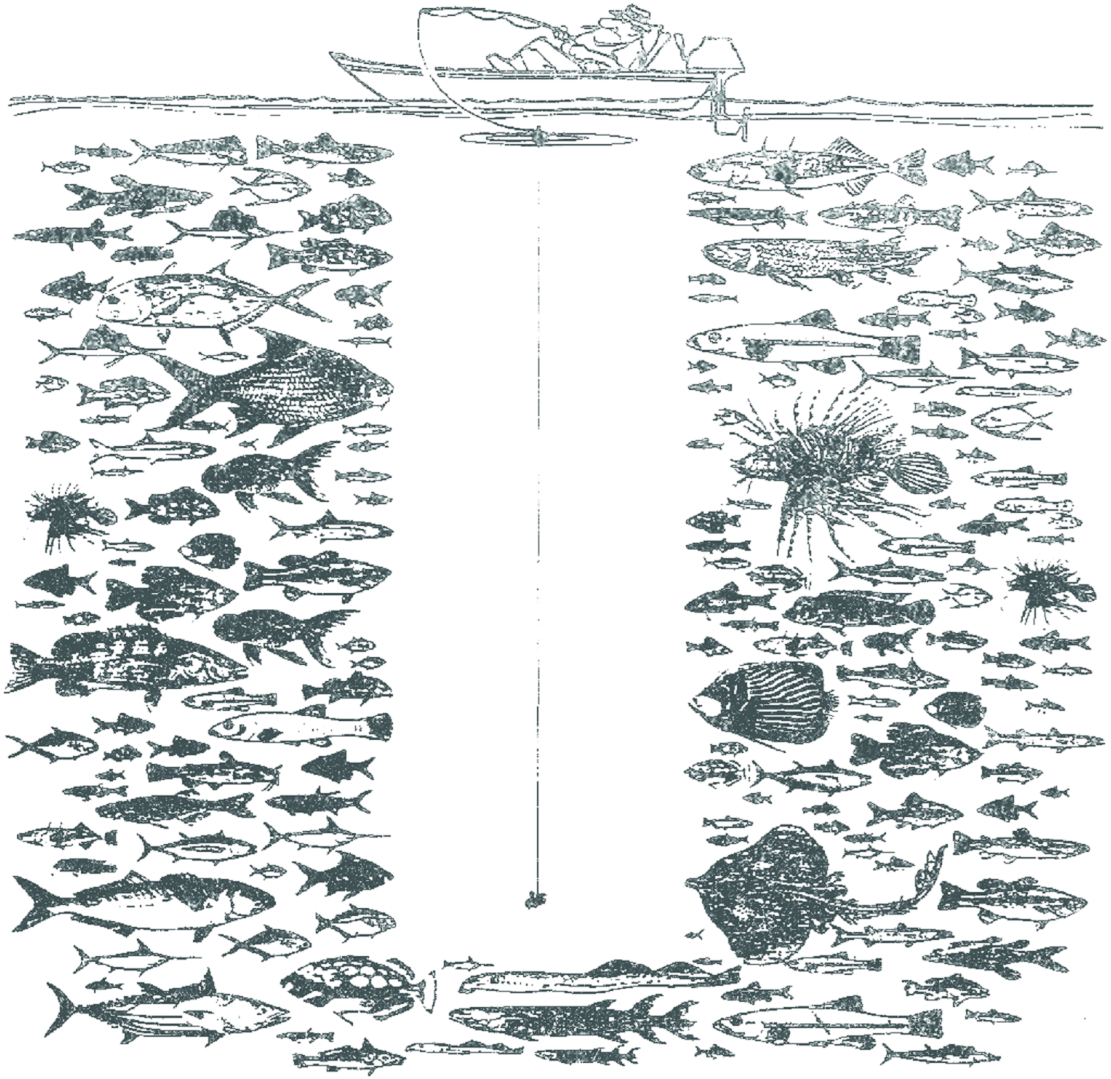
lecture

active learning

study

exam

Programmeren = software testen

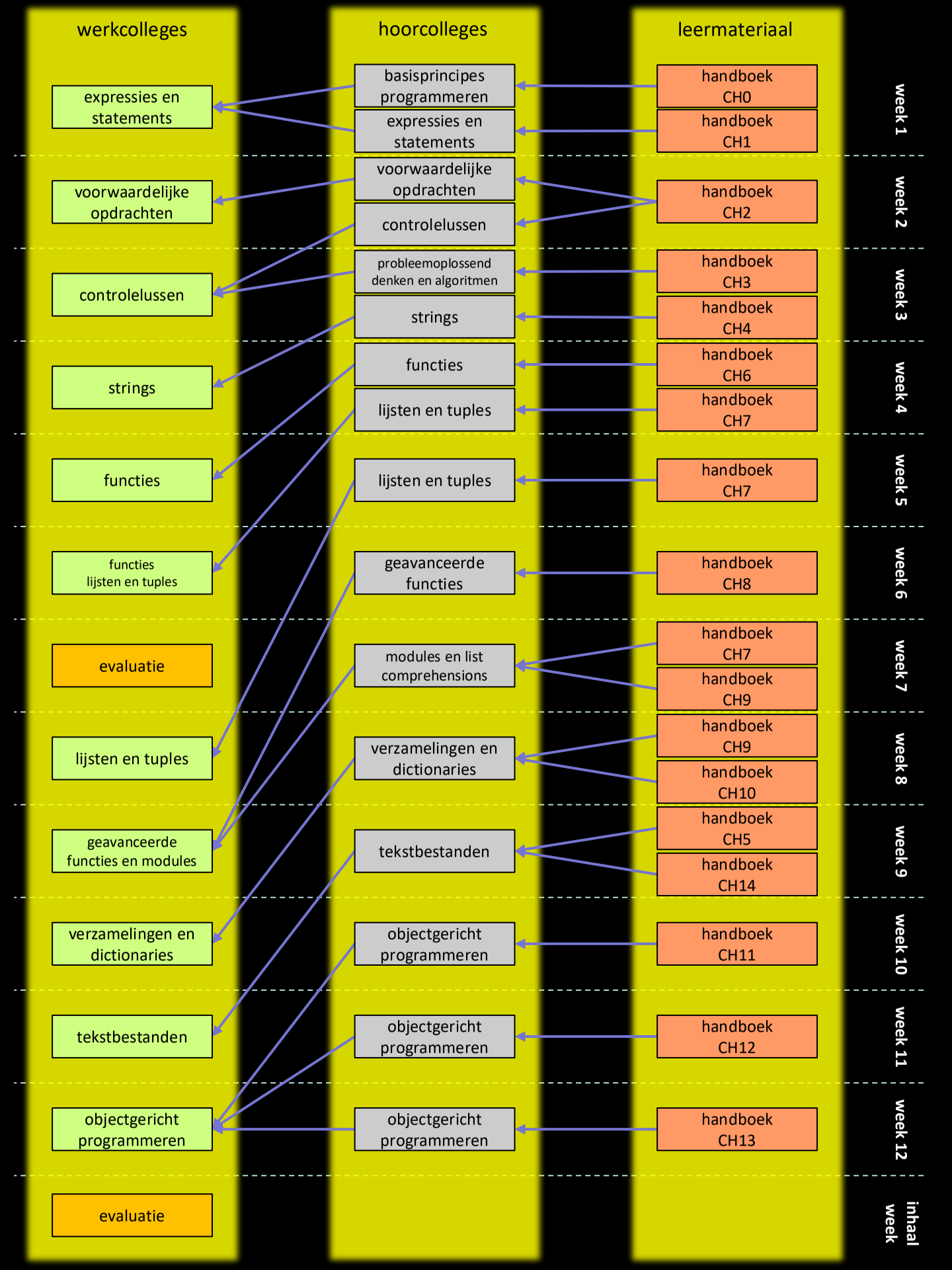


UNIVERSITEIT GENT

Evaluaties

- opgelegde oefeningen
  - evaluatiereeks = 5 reeksen van 6 verplichte oefeningen
  - verplichten studenten om theorie om te zetten in de praktijk
  - wekelijkse deadlines op dinsdag om 22:00
  - indienen via Dodona vóór vastgelegde deadlines (zie overzicht)
  - bouwen geleidelijk op naar niveau examenoefeningen
- niet periodegebonden evaluaties
  - 2 evaluaties in weken 7 en 13 (tijdens werkcolleges)
  - per evaluatie 2 oefeningen op te lossen binnen 2 uur
  - niveau: evaluatie-oefeningen zijn makkelijker dan examenoefeningen
  - testen beheersen van basisvaardigheden programmeren
- periodegebonden evaluaties (examen)
  - examen = 3 oefeningen op te lossen binnen 3.5 uur
  - niveau: examenoefeningen zijn moeilijker dan evaluatie-oefeningen

UNIVERSITEIT GENT

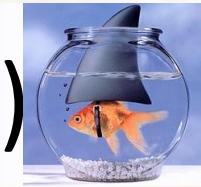

werkcolleges
hoorcolleges
leermateriaal
expressies en statements
voorwaardelijke opdrachten
controlelussen
strings
functies
functies lijsten en tuples
evaluatie
lijsten en tuples
geavanceerde functies en modules
verzamelingen en dictionaries
tekstbestanden
objectgericht programmeren
evaluatie
basisprincipes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

Evaluaties

- opgelegde oefeningen
  - evaluatiereeks = 5 reeksen van 6 verplichte oefeningen
  - verplichten studenten om theorie om te zetten in de praktijk
  - wekelijkse deadlines op dinsdag om 22:00
  - indienen via Dodona vóór vastgelegde deadlines (zie overzicht)
  - bouwen geleidelijk op naar niveau examenoefeningen
- niet periodegebonden evaluaties
  - 2 evaluaties in weken 7 en 13 (tijdens werkcolleges)
  - per evaluatie 2 oefeningen op te lossen binnen 2 uur
  - niveau: evaluatie-oefeningen zijn makkelijker dan examenoefeningen
  - testen beheersen van basisvaardigheden programmeren
- periodegebonden evaluaties (examen)
  - examen = 3 oefeningen op te lossen binnen 3.5 uur
  - niveau: examenoefeningen zijn moeilijker dan evaluatie-oefeningen

UNIVERSITEIT GENT

Leren programmeren = oefenen (3x)



Evaluaties

- niet-periodegebonden evaluaties tellen mee voor 4/20
  - driemaal winst !!!
  - niet herneembaar tijdens tweede zittijd
- andere oefeningen
  - vrijblijvend (bv. voorbereiding examen)
  - examenvragen van vorige jaargangen
  - leren programmeren = oefenen, oefenen, oefenen, …

UNIVERSITEIT GENT

Puntenberekening evaluaties

opgelegde oefeningen

c : aantal opgelegde oefeningen dat correct werd ingediend tegen wekelijkse deadlines

a : aantal opgelegde oefeningen

evaluatie-oefeningen

s: score behaald voor de evaluatie-oefeningen

UNIVERSITEIT GENT

$$\text{finale score} = s \times \frac{c}{a}$$

Puntenberekening evaluaties

opgelegde oefeningen

c : aantal opgelegde oefeningen dat correct werd ingediend tegen wekelijkse deadlines

a : aantal opgelegde oefeningen

30/30

evaluatie-oefeningen

s: score behaald voor de evaluatie-oefeningen

16/20

$$\text{finale score} = s \times \frac{c}{a} = 16 \times \frac{30}{30} = 16$$

UNIVERSITEIT GENT

Puntenberekening evaluaties

opgelegde oefeningen

c : aantal opgelegde oefeningen dat correct werd ingediend tegen wekelijkse deadlines

a : aantal opgelegde oefeningen

18/30

evaluatie-oefeningen

s: score behaald voor de evaluatie-oefeningen

16/20

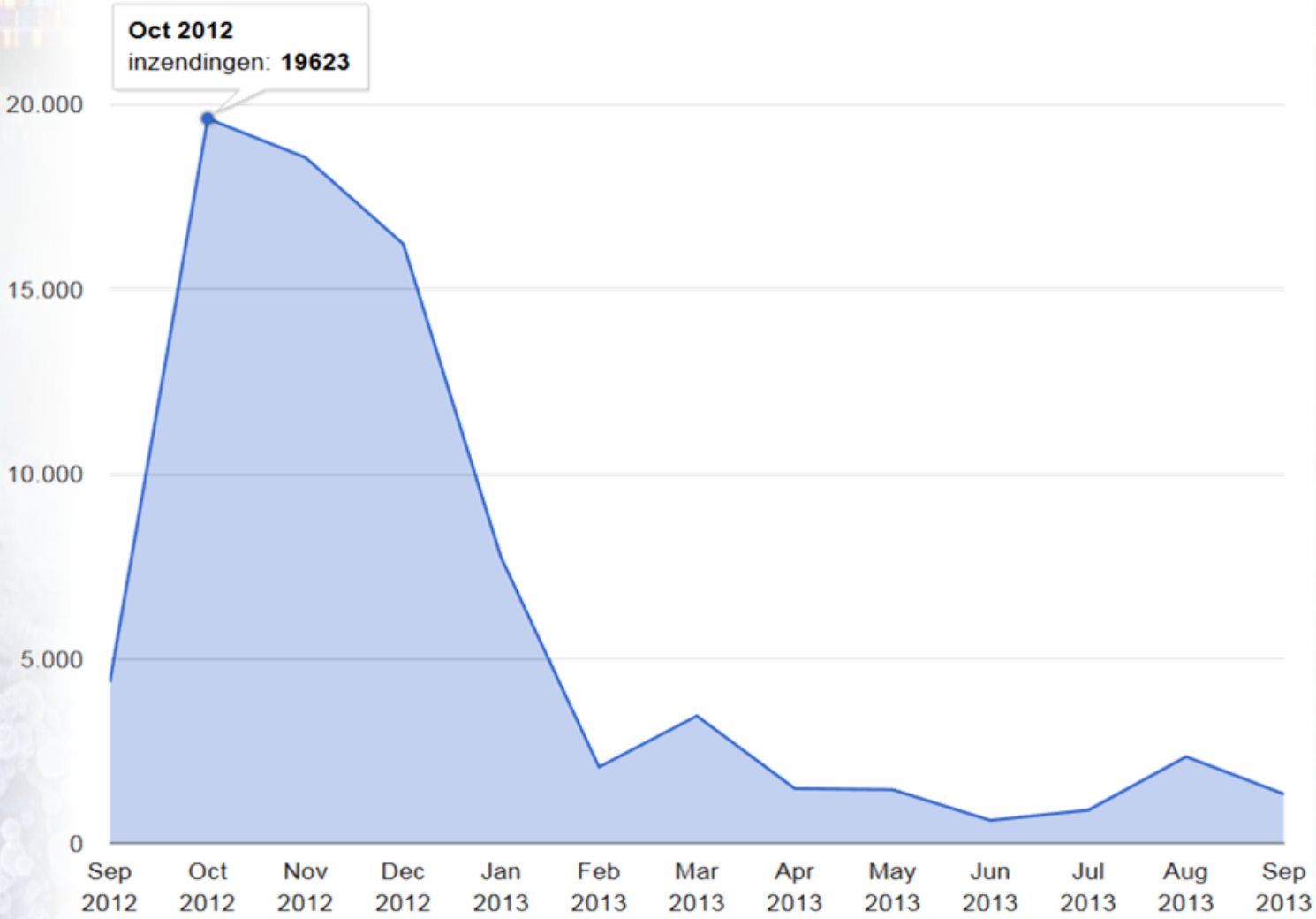
$$\text{finale score} = s \times \frac{c}{a} = 16 \times \frac{18}{30} = 9.6$$

UNIVERSITEIT GENT

Automatische feedback

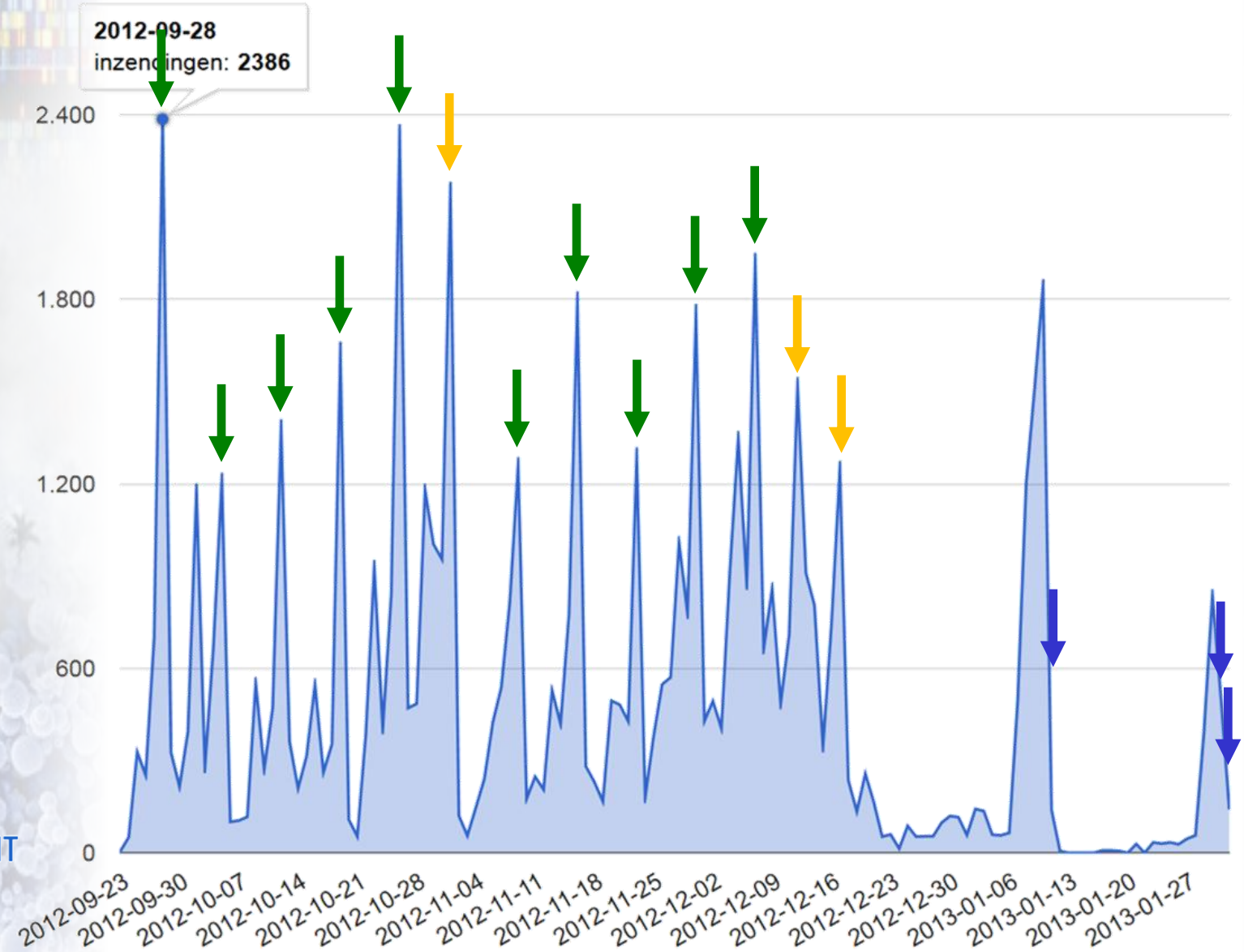
- opleidingsonderdeel: programmeren (Python)
  - 10 oefeningenreeksen
  - 822 oefeningen
  - 483 studenten
  - 66 495 correct opgeloste oefeningen
  - 339 466 ingezonden oplossingen
  - 9 878 833 regels broncode
  - 287 513 007 broncodekarakters

Datavolume



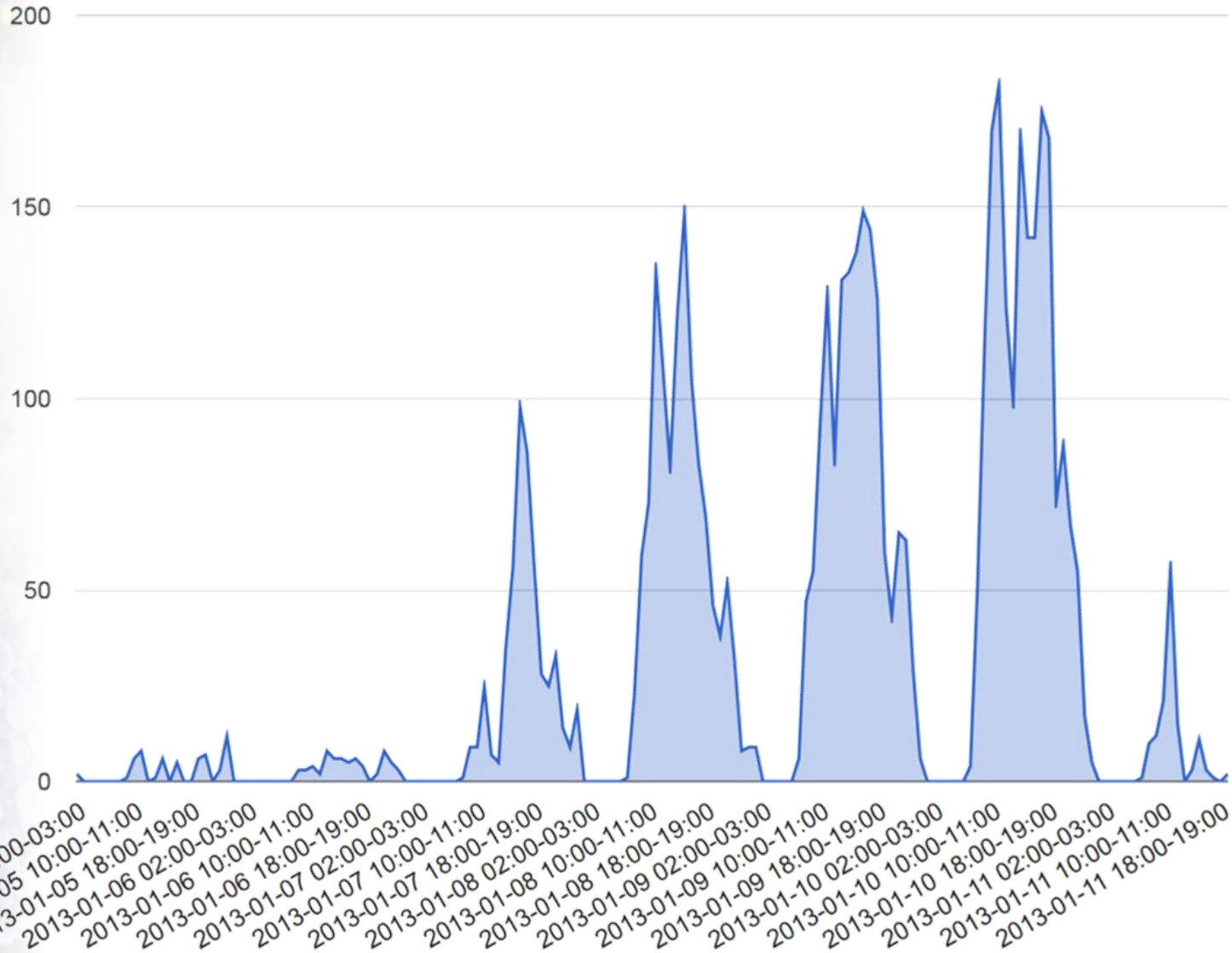
UNIVERSITEIT GENT

Datavolume



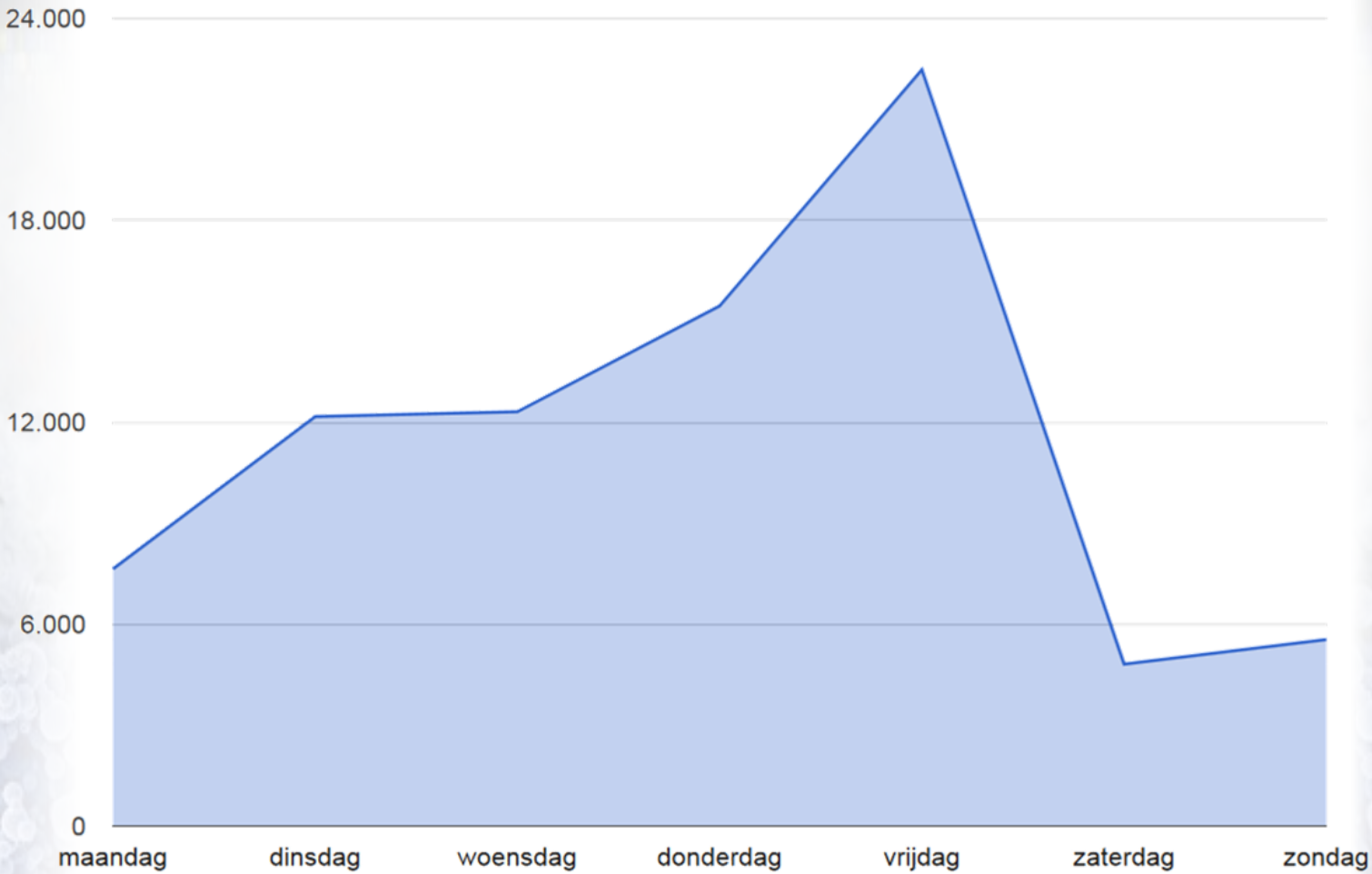
UNIVERSITEIT GENT

Datavolume



UNIVERSITEIT GENT

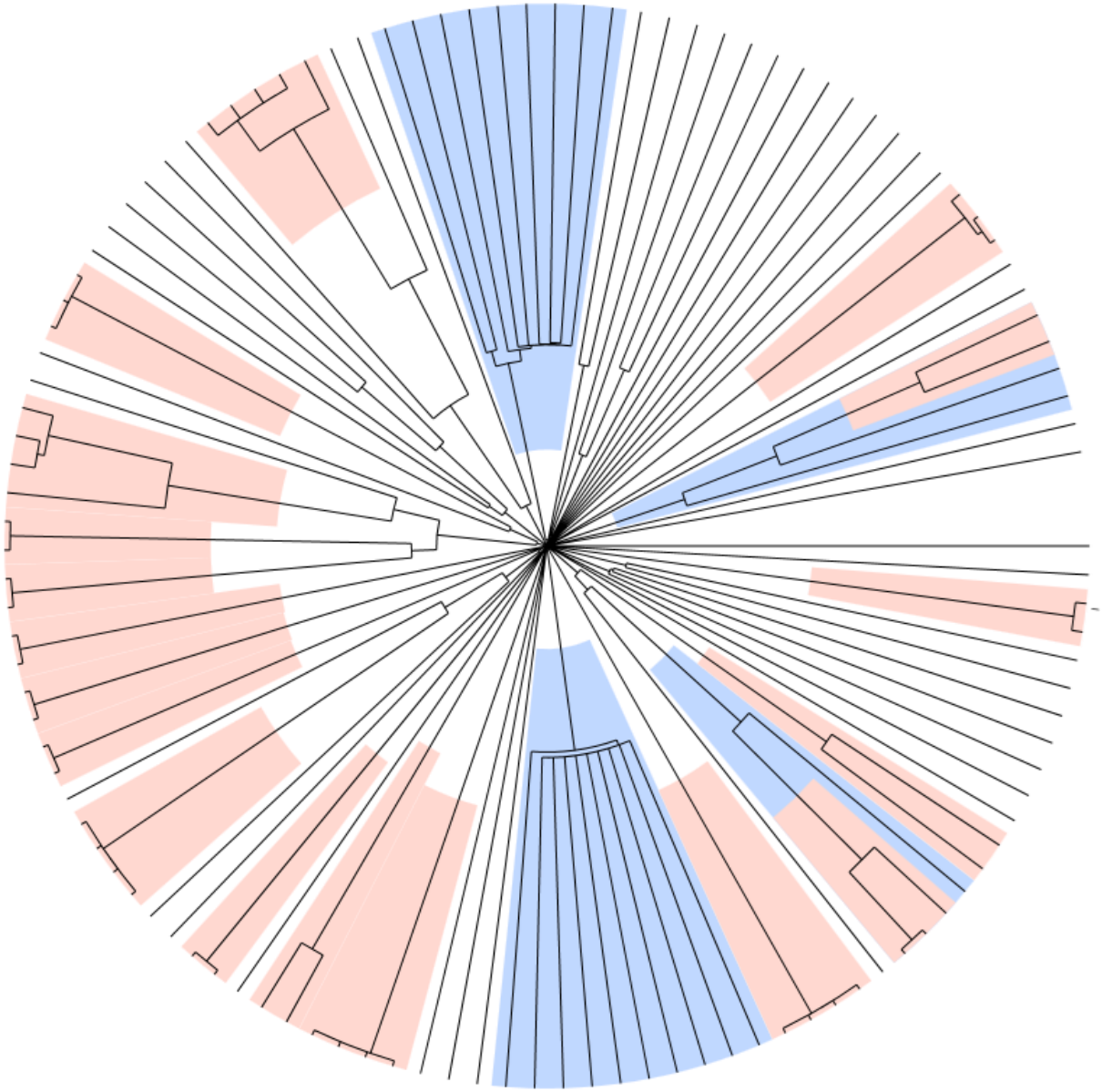
Datavolume



UNIVERSITEIT GENT

Plagiaat

Plagiaatdetectie



UNIVERSITEIT GENT

Waarden en gegevenstypes

Waarden en gegevenstypes

- **object**: fundamenteel ding (zoals een letter of getal) dat door computerprogramma gemanipuleerd wordt
  - strings: `'Hello, World!'`
  - integers: `17`
  - floats: `3.2`

```
>>> print(4)
4
>>> print(2.7)
2.7
```

UNIVERSITEIT GENT

Waarden en gegevenstypes

- **object**: fundamenteel ding (zoals een letter of getal) dat door computerprogramma gemanipuleerd wordt
  - strings: `'Hello, World!'`
  - integers: `17`
  - floats: `3.2`
- elk object heeft een gegevenstype (*strongly typed*)

```
>>> type('Hello, World!')          # strings
<class 'str'>
>>> type(17)                       # gehele getallen
<class 'int'>
>>> type(3.2)                      # reële getallen
<class 'float'>
```

UNIVERSITEIT GENT

Waarden en gegevenstypes

- **object**: fundamenteel ding (zoals een letter of getal) dat door computerprogramma gemanipuleerd wordt
  - strings: `'Hello, World!'`
  - integers: `17`
  - floats: `3.2`
- elk object heeft een gegevenstype (*strongly typed*)

```
>>> type('17')
<class 'str'>
>>> type('3.2')
<class 'str'>
```

UNIVERSITEIT GENT

Waarden en gegevenstypes

- strings staan tussen enkele of dubbele aanhalingstekens
  - strings tussen dubbele aanhalingstekens kunnen enkele aanhalingstekens bevatten
  - strings tussen enkele aanhalingstekens kunnen dubbele aanhalingstekens bevatten

```
>>> type('Dit is een string.')
<class 'str'>
>>> type("En dit ook.")
<class 'str'>
>>> print("Bruce's beard")
Bruce's beard
>>> print('The knights who say "Ni!"')
The knights who say "Ni!"
```

UNIVERSITEIT GENT

Waarden en gegevenstypes

- gebruik geen komma's of punten om duizendtallen te scheiden bij grote gehele getallen (Amerikaanse notatie)
  - 1,000,000 wordt geïnterpreteerd als *tuple* van 3 objecten die elk afzonderlijk moeten uitgeschreven worden

```
>>> print(1,000,000)
1 0 0
>>> print('a', 'b', 'c')
a b c
>>> print(1.000.000)
  File "<stdin>", line 1
    print(1.000.000)
                  ^
SyntaxError: invalid syntax
```

UNIVERSITEIT GENT

Variabelen

- **variabele**: naam die verwijst naar object
  - toekenningsopdracht (*assignment statement*)
    - variabele aanmaken
    - variabele laten verwijzen naar object
  - verwijst naar plaats waar object wordt opgeslaan in werkgeheugen van computer
  - naam gebruiken in expressie haalt object terug op uit werkgeheugen

```
>>> welkom = 'Hello, World!'
>>> print(welkom)
Hello, World!
```

werkgeheugen (RAM)

UNIVERSITEIT GENT

Variabelen

- toekenningsoperator =
  - gebruikt in toekenningsopdracht
  - niet te verwarren met gelijkheidstest (==)
  - kent *object* (rechterlid) toe aan *naam* (linkerlid)
    - linkerlid moet dus altijd bestaan uit variabelenaam
    - rechterlid moet altijd expressie zijn die object oplevert

```
>>> boodschap = "What's up, Doc?"
>>> n = 17
>>> pi = 3.14159
>>> 17 = n
  File "<stdin>", line 1
SyntaxError: can't assign to literal
```

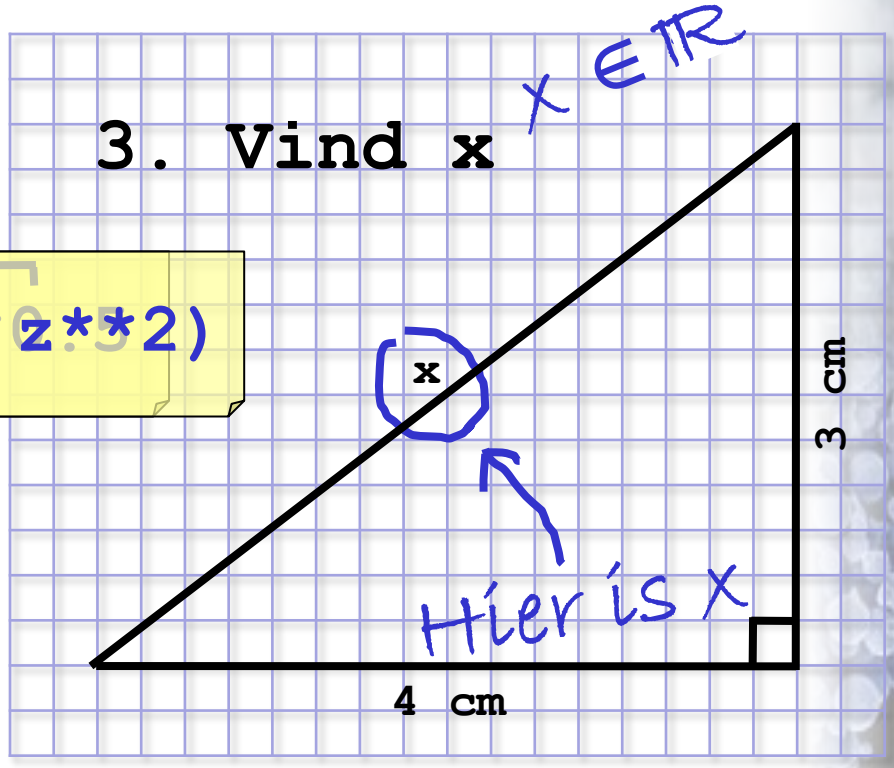
UNIVERSITEIT GENT

Variabelen

- **variabele**: naam die verwijst naar object
  - naam en object zijn twee verschillende componenten
  - gegevenstype is eigenschap van object
    - kan via naam opgevraagd worden

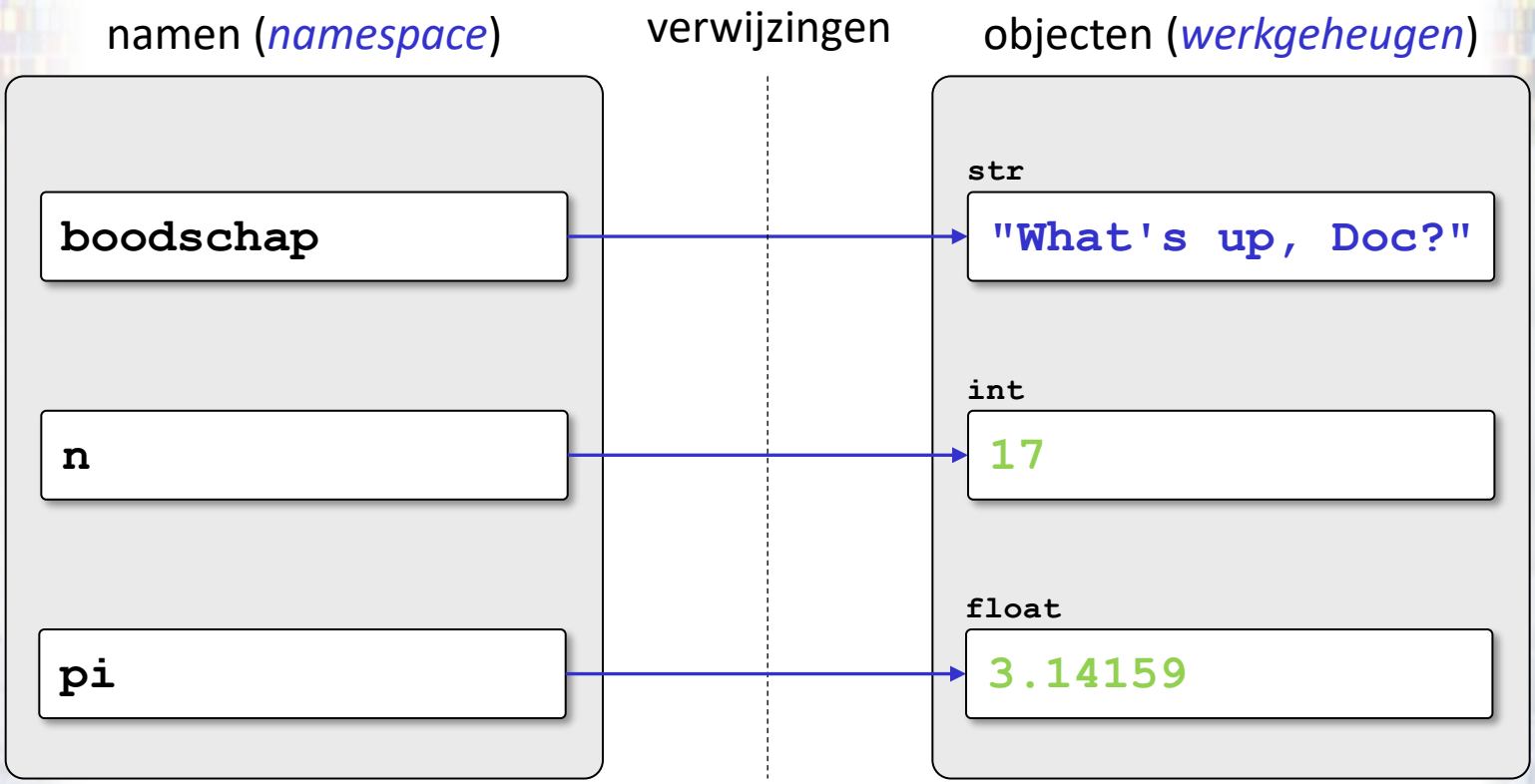
```
>>> type(boodschap)
<class 'str'>
>>> type(n)
<class 'int'>
>>> type(pi)
<class 'float'>
```

```
x = math.sqrt(y**2 + z**2)
```



UNIVERSITEIT GENT

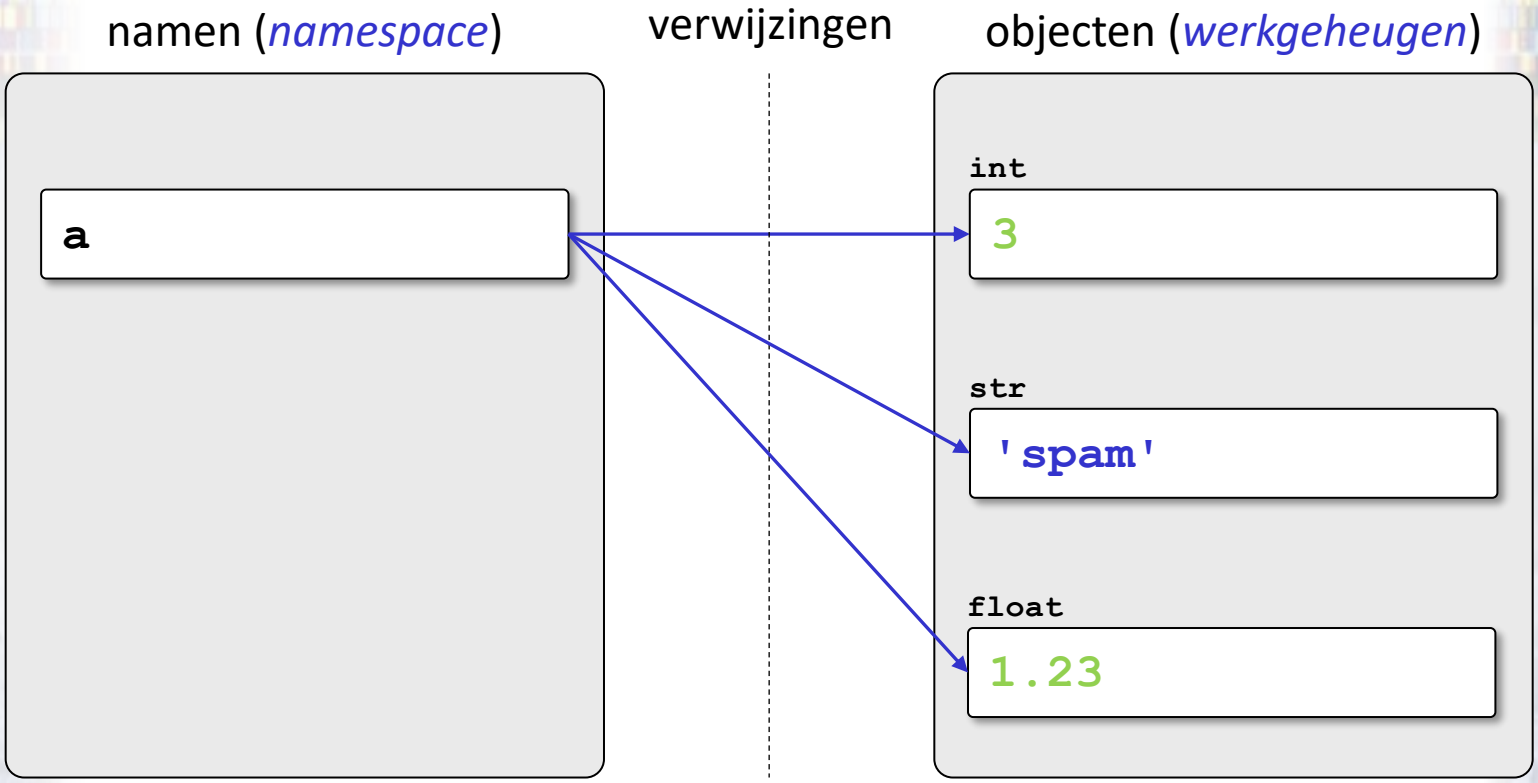
Toestandsdiagram




```
>>> boodschap = "What's up, Doc?"
>>> n = 17
>>> pi = 3.14159
```

UNIVERSITEIT GENT

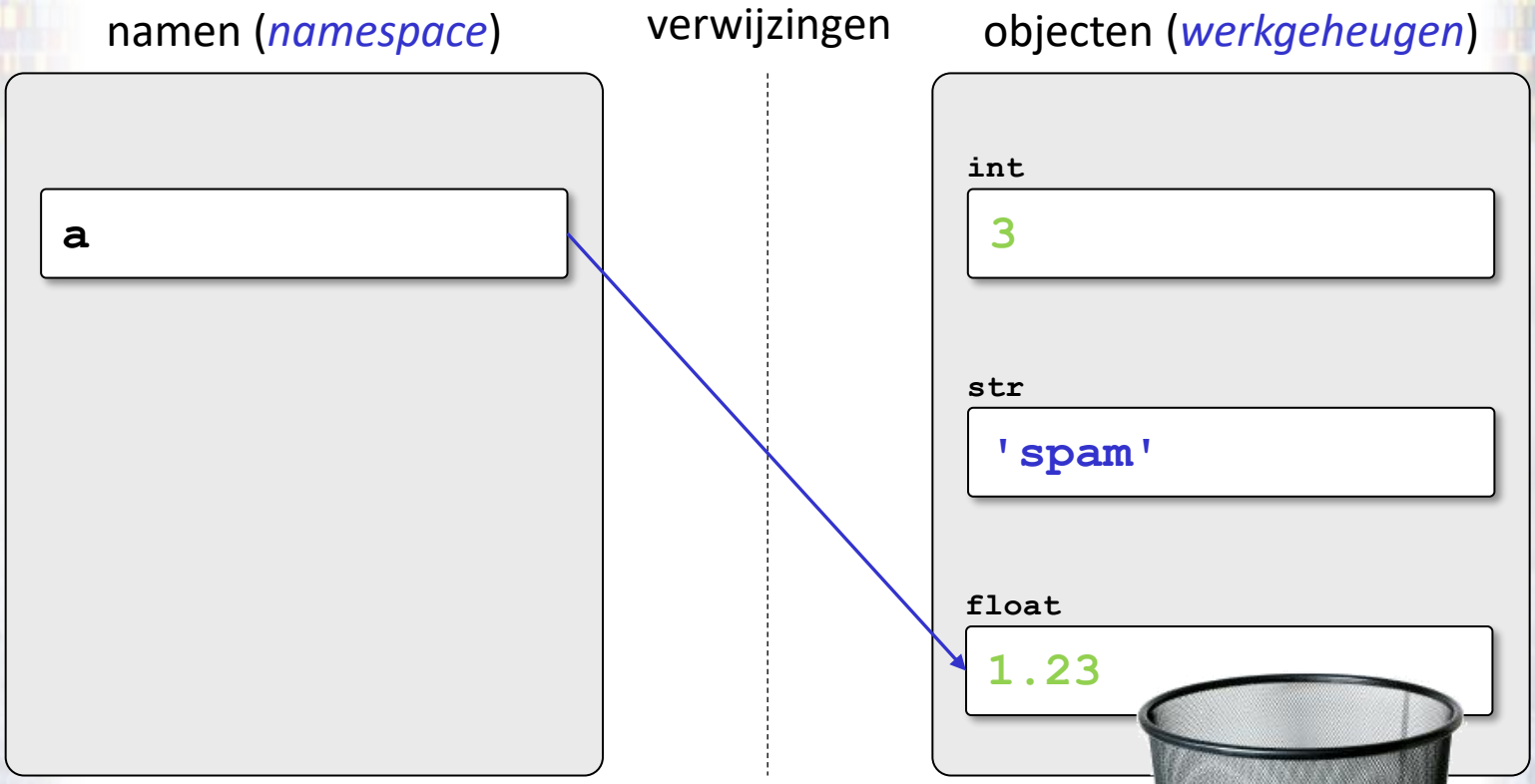
Dynamic typing




```
>>> a = 3          # een geheel getal
>>> a = 'spam'     # nu is het een string
>>> a = 1.23       # en nu een floating point getal
```

UNIVERSITEIT GENT

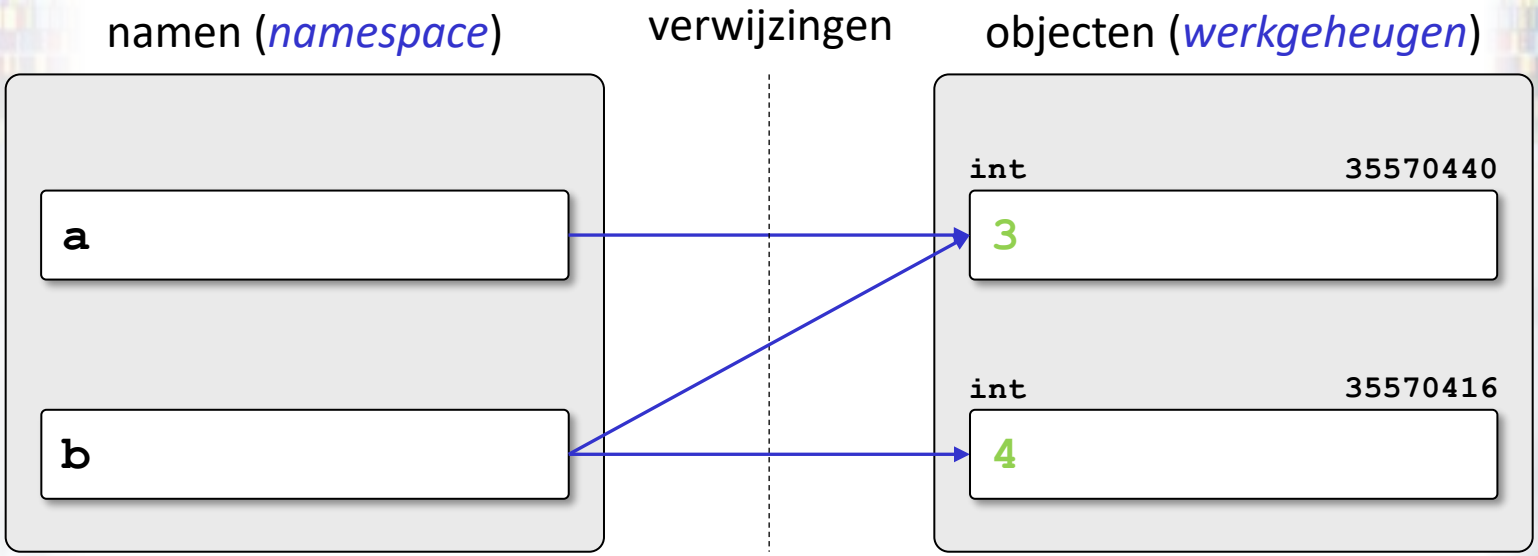
Garbage collection




```
>>> a = 3          # een geheel getal
>>> a = 'spam'     # nu is het een string
>>> a = 1.23       # en nu een f
```

UNIVERSITEIT GENT

Meerdere verwijzingen




```
>>> a = 3
>>> b = 4
>>> print(id(a), id(b))
35570440 35570416
>>> b = a
>>> print(id(a), id(b))
35570440 35570440
```

UNIVERSITEIT GENT

Programmeren met stijl

stijlregel

gebruik lange (informatieve) namen die betekenis van variabelen documenteren

Variabelen benoemen

- namen van variabelen zijn onbeperkt in lengte
  - moeten beginnen met een letter
  - bestaan enkel uit letters, cijfers en underscores (_)
  - case sensitive: verschil tussen hoofdletters en kleine letters
    - **spam** en **SPAM** staan voor twee verschillende variabelen

```
>>> 3miljard = 'drieduizend miljoen'
SyntaxError: invalid syntax
>>> googol$ = 10000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
SyntaxError: invalid syntax
```

Programmeren met stijl

stijlregel

gebruik voor variabelen geen namen
die enkel bestaan uit hoofdletters

Variabelen benoemen

- namen van variabelen zijn onbeperkt in lengte
  - moeten beginnen met een letter
  - bestaan enkel uit letters, cijfers en underscores (_)
  - case sensitive: verschil tussen hoofdletters en kleine letters
    - **`spam`** en **`SPAM`** staan voor twee verschillende variabelen
  - Python sleutelwoorden kunnen niet gebruikt worden
    - definiëren regels en structuur van programmeertaal

```
>>> 3miljard = 'drieduizend miljoen'
SyntaxError: invalid syntax
>>> googol$ = 1000000000…
SyntaxError: invalid syntax
>>> class = "Programmeren"
SyntaxError: invalid syntax
```

UNIVERSITEIT GENT

Python sleutelwoorden

False	None	True	and	as	assert
break	class	continue	def	del	elif
else	except	finally	for	from	global
if	import	in	is	lambda	nonlocal
not	or	pass	raise	return	try
while	with	yield			

UNIVERSITEIT GENT

Statements

- **statement**: instructie die Python kan uitvoeren
  - voorbeelden
    - printopdracht (*print functie*)
    - toekenningsopdracht (*assignment statement*)
  - Python shell
    - voert statement uit en toont resultaat (indien er één is)
    - resultaat van printopdracht is waarde
    - toekenningsopdracht produceert geen resultaat

```
>>> print(1)
1
>>> x = 2
>>> print(x)
2
```

UNIVERSITEIT GENT

Statements

- **statement**: instructie die Python kan uitvoeren
  - voorbeelden
    - printopdracht (*print functie*)
    - toekenningsopdracht (*assignment statement*)
  - Python script
    - bevat doorgaans reeks van statements
    - resultaten verschijnen één na één terwijl statements uitgevoerd worden

statements.py

```
print(1)
x = 2
print(x)
```

```
$ python3 statements.py
1
2
$
```

UNIVERSITEIT GENT

Dodona

veelgemaakte fout

in Dodona moet een Python script ingediend worden als oplossing voor een opgave, geen kopie van shell sessie

Evaluatie van expressies

- **expressie**: combinatie van objecten, variabelen en operatoren
  - evaluatie van expressie resulteert in object
    - expressie kan in rechterlid van toekenningsopdracht voorkomen
    - Python shell toont resultaat van evaluatie
    - object op zichzelf is ook expressie
    - variabele op zichzelf is ook expressie

```
>>> 1 + 1
2
>>> 17
17
>>> x
2
```

UNIVERSITEIT GENT

Evaluatie van expressies

- evaluatie van expressie ≠ waarde van object afdrukken
  - Python shell
    - toont resultaat van expressie in formaat dat je zou gebruiken om object in broncode voor te stellen
    - **`print`** functie toont letterlijke waarde van expressie

```
>>> boodschap = "What's up, Doc?"
>>> boodschap
"What's up, Doc?"
>>> print(boodschap)
What's up, Doc?
```

UNIVERSITEIT GENT

Evaluatie van expressies

- evaluatie van expressie ≠ waarde van object afdrukken
  - Python shell
    - toont resultaat van expressie in formaat dat je zou gebruiken om object in broncode voor te stellen
    - **`print`** functie toont letterlijke waarde van expressie
  - Python script
    - expressie op zichzelf is geldig statement
    - expressie op zichzelf produceert geen resultaat

expressies.py

```
17
3.2
'Hello, World!'
1 + 1
```

```
$ python3 expressies.py
$
```

Hoe zou je dit script aanpassen zodat het de waarde van de vier expressies uitschrijft ?

UNIVERSITEIT GENT

Operatoren

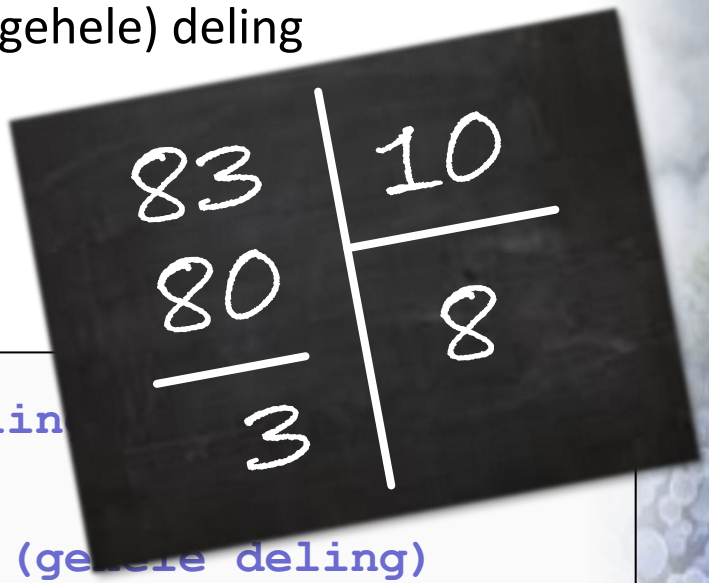
- **operator**: speciaal symbool dat bewerking voorstelt
  - voorbeelden
    - optelling: **+**
    - vermenigvuldiging: *****
    - machtsverheffing: ******
  - **operandi**: waarden waarop operator inwerkt
  - naam van variabele als operand wordt vervangen door gerefereerde object vóór expressie geëvalueerd wordt

```
20 + 32
uren - 1
uren * 60 + minuten
minuten // 60
5 ** 2
(5 + 9) * (15 - 7)
```

UNIVERSITEIT GENT

Deling

- Python onderscheidt **reële deling** van **gehele deling**
  - operator **/** berekent reële deling (levert altijd float op)
  - operator **//** berekent quotiënt (gehele deling)
  - modulo operator **%** berekent rest na (gehele) deling




```
>>> 83 / 10          # reële delin
8.3
>>> 83 // 10         # quotiënt (ge    eling)
8
>>> 83 % 10          # rest na deling (modulo)
3
```

UNIVERSITEIT GENT

Deling

- Python onderscheidt **reële deling** van **gehele deling**
  - operator **/** berekent reële deling (levert altijd float op)
  - operator **//** berekent quotiënt (gehele deling)
  - modulo operator **%** berekent rest na (gehele) deling

```
>>> 83.0 / 10.0     # reële deling
8.3
>>> 83.0 // 10      # quotiënt (gehele deling)
8.0
>>> 83 // 10.0      # quotiënt (gehele deling)
8.0
>>> 83.0 % 10       # rest na deling (modulo)
3.0
>>> 83 % 10.0       # rest na deling (modulo)
3.0
```

UNIVERSITEIT GENT

Typeconversies

- elk Python gegevenstype heeft ingebouwde functie die argument omzet naar corresponderende gegevenstype
  - **`int(`*`argument`*`)`** zet argument om naar gegevenstype **`int`**
  - fout tijdens uitvoering indien argument niet kan omgezet worden

```
>>> float(83) // 10
8.0
>>> int('32')
32
>>> int('Hallo')
ValueError: invalid literal for int() with base 10
```

UNIVERSITEIT GENT

Typeconversies

- elk Python gegevenstype heeft ingebouwde functie die argument omzet naar corresponderende gegevenstype
  - **`int(`*`argument`*`)`** zet argument om naar gegevenstype **`int`**
  - fout tijdens uitvoering indien argument niet kan omgezet worden
  - floats worden afgekapt bij conversie naar int

```
>>> int(-2.3)
-2
>>> int(3.99999)
3
>>> int('42')
42
>>> int(1.0)
1
```

UNIVERSITEIT GENT

Typeconversies

- elk Python gegevenstype heeft ingebouwde functie die argument omzet naar corresponderende gegevenstype
  - **`float(`*`argument`*`)`** zet argument om naar gegevenstype **`float`**
  - verschil tussen integer **`1`** en floating-point **`1.0`**
    - verschillend gegevenstype
    - verschillende interne voorstelling binnen computersysteem (zie extra deel presentatie)

```
>>> float(32)
32.0
>>> float('3.14159')
3.14159
>>> float(1)
1.0
```

UNIVERSITEIT GENT

Typeconversies

- elk Python gegevenstype heeft ingebouwde functie die argument omzet naar corresponderende gegevenstype
  - **`str(`*`argument`*`)`** zet argument om naar gegevenstype **`str`**

```
>>> str(32)
'32'
>>> str(3.14149)
'3.14149'
>>> str(True)
'True'
>>> str(true)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'true' is not defined
```

UNIVERSITEIT GENT

Evaluatie van expressies

- evaluatie van expressies gebeurt volgens voorrangsregels
  - hoe hoger operator in tabel, hoe groter zijn voorrang
  - operatoren op zelfde niveau worden vlnr geëvalueerd
  - volledige voorrangstabel in Appendix E in handboek

operator	omschrijving
`()`	haakjes (groeperen)
`**`	machtsverheffing
`+x`, `-x`	positief, negatief (unair)
`*`, `/`, `%`	vermenigvuldiging, deling, rest
`+`, `-`	optelling, aftrekking

UNIVERSITEIT GENT

Stringoperatoren

- rekenen met strings lukt doorgaans niet
  - zelfs al zien de strings er als getallen uit
  - Python doet geen impliciete typeconversies
    - behalve tussen numerieke types: int, float, complex, …
    - gemengde numerieke bewerkingen: int → float → complex

```
>>> boodschap = "What's up, Doc?"
>>> boodschap - 1
TypeError: unsupported operand type(s) for -: 'str' and 'int'
>>> 'Hallo' / 123
TypeError: unsupported operand type(s) for /: 'str' and 'int'
>>> boodschap * 'Hallo'
TypeError: can't multiply sequence by non-int of type 'str'
>>> '15' + 2
TypeError: cannot concatenate 'str' and 'int' objects
```

UNIVERSITEIT GENT

Stringoperatoren

- rekenen met strings lukt doorgaans niet
  - twee uitzonderingen:
    - strings optellen voegt strings samen (*concatenation*)
    - string vermenigvuldigen met integer herhaalt string (*repetition*)
      - één operand moet string zijn en andere operand moet **`int`** zijn
    - let op analogie
      - **`4 * 3 == 4 + 4 + 4`**
      - **`'Spam' * 3 == 'Spam' + 'Spam' + 'Spam'`**

```
>>> fruit = 'appel'
>>> gebak = 'taart'
>>> print(fruit + gebak)
appeltaart
>>> 'Spam' * 3
'SpamSpamSpam'
>>> 3 * 'Spam'
'SpamSpamSpam'
```

UNIVERSITEIT GENT

Stringoperatoren

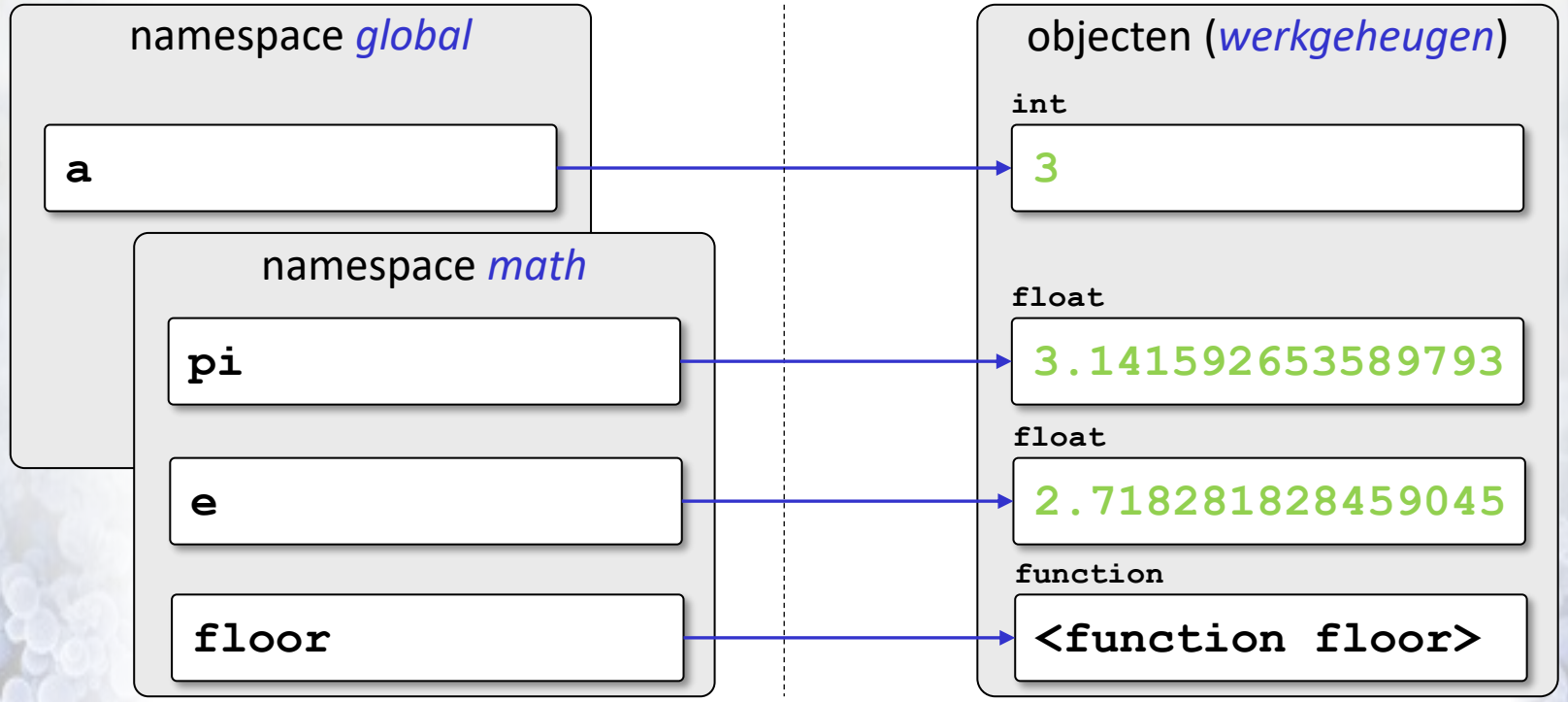
- bijkomende bewerkingen
  - zie ingebouwde **`help`**-functie
  - zie Python documentatie online

```
>>> help(str)
Help on class str in module __builtin__:
...
>>> boodschap = "What's up, Doc?"
>>> help(boodschap.upper)
Help on built-in function upper:

upper(...)
    S.upper() -> string

    Return a copy of the string S converted to uppercase.
>>> print(boodschap.upper())
WHAT'S UP, DOC?
```

UNIVERSITEIT GENT

Python uitbreiden

- standaardtaal kan uitgebreid worden met modules
  - voorbeeld: **`math`** module




```
>>> a = 3
>>> import math
```

UNIVERSITEIT GENT

Python uitbreiden

- standaardtaal kan uitgebreid worden met modules
  - voorbeeld: **`math`** module

```
>>> import math
>>> help(math)
Help on built-in module math:
...
>>> help(math.floor)
Help on built-in function floor in module math:

floor(...)
    floor(x)

    Return the floor of x as a float.
    This is the largest integral value <= x.
>>> math.floor(3.7)
3
```

UNIVERSITEIT GENT

Python uitbreiden

- standaardtaal kan uitgebreid worden met modules
  - voorbeeld: **`math`** module

```
>>> math.floor(3.7)          # afronden naar beneden
3
>>> int(3.7)                 # afkappen
3
>>> round(3.7)               # afronden
4
>>> math.floor(-3.7)         # afronden naar beneden
-4
>>> int(-3.7)                # afkappen
-3
>>> round(-3.7)              # afronden
-4
>>> math.ceil(-3.7)          # afronden naar boven
-3
```

UNIVERSITEIT GENT

Gegevens invoeren

- ingebouwde functie voor invoer via toetsenbord
  - **`input(`*`argument`*`)`**
    - invoer wordt als string teruggegeven
    - omzetten naar specifiek gegevenstype met conversiefuncties
    - evaluatie van expressie met ingebouwde functie **`eval()`**

```
>>> n = input('Please enter your name: ')
Please enter your name: Arthur, King of the Britons
>>> print(n)
Arthur, King of the Britons
>>> print(int(input('Square root of 1764 = ')))
Square root of 1764 = 42
42
>>> n = eval(input('Enter a numerical expression: '))
Enter a numerical expression: 3 * 7
>>> print(n)
21
```

UNIVERSITEIT GENT

Dodona

veelgemaakte fout

de functie `input()` geeft een string als resultaat terug, en meestal moet deze string omgezet worden naar een specifiek gegevenstype aan de hand van een conversiefunctie

Samenstelling

- elementen van programma werden afzonderlijk besproken
  - variabelen, expressies en statements
  - bouwstenen kunnen onbeperkt samengesteld worden (*composition*)
    - waarde van expressie afdrukken
    - willekeurige expressie in rechterlid van toekenningsopdracht

```
>>> print(17 + 3)
20
>>> uren = 20; minuten = 32
>>> print('Aantal minuten sedert middernacht:', \
...       uren * 60 + minuten)
Aantal minuten sedert middernacht: 1232
>>> percentage = (minuten * 100) // 60
>>> print(percentage)
53
```

UNIVERSITEIT GENT

Programmeren met stijl

context

naargelang programma's groter en complexer worden, worden ze ook steeds moeilijker te lezen; formele talen zijn compact, en het is vaak moeilijk om naar een stuk code te kijken en uit te vissen wat het doet, of waarom

stijlregel

voeg aan je broncode commentaar toe die in natuurlijke taal uitlegt wat het programma doet

Commentaar

- commentaar loopt van hekje (#) tot einde regel
- commentaar wordt genegeerd door Python
  - boodschap is bedoeld voor wie broncode in toekomst moet lezen
- afspraak examen
  - grote lijnen broncode moeten duidelijk zijn uit commentaar
  - anders wordt broncode niet verder verbeterd door examinator

commentaar.py

```
minuten = int(input('Aantal minuten: '))

# bereken percentage van het uur dat voorbij is
percentage = (minuten * 100) // 60  # gehele deling!

print(str(percentage) + '% verstreken')
```

UNIVERSITEIT GENT

Dodona

tip

gebruik Q&A module in Dodona om vragen te stellen over een opgave uit Dodona

Huiswerk

- afwerken opgelegde oefeningen reeks 01 (deadline dinsdag 30 september 2025, 22:00)
  - ISBN (demo, video)
  - Som van twee getallen
  - Examens verbeteren
  - The pudding guy
  - Ontwikkelingsindex
  - De gestopte klok

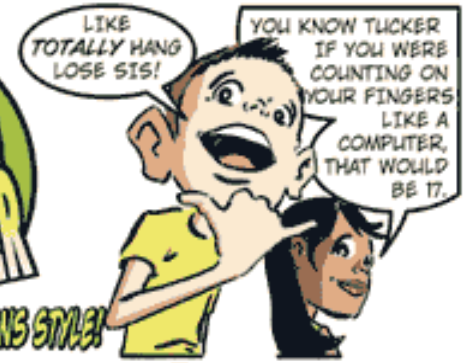
Huiswerk (volgende les)

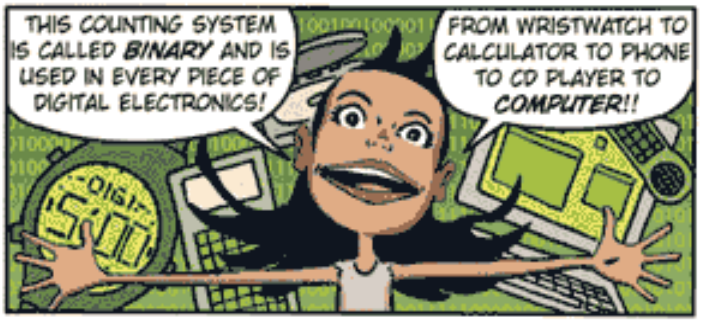
- bekijk stijlregels van Python
- lees hoofdstuk 2 van handboek (controlestructuren)
- leer online help te gebruiken
  - gebruik help-functie om math module te bestuderen
  - Python Standard Library: Python uitbreidingen
- bestudeer extra slides over interne voorstelling van waarden
- voorbeeldoefeningen volgende les (reeks 02)
  - Body-mass index
  - Babysit
  - Botsing

UNIVI GENT

COUNT LIKE A COMPUTER

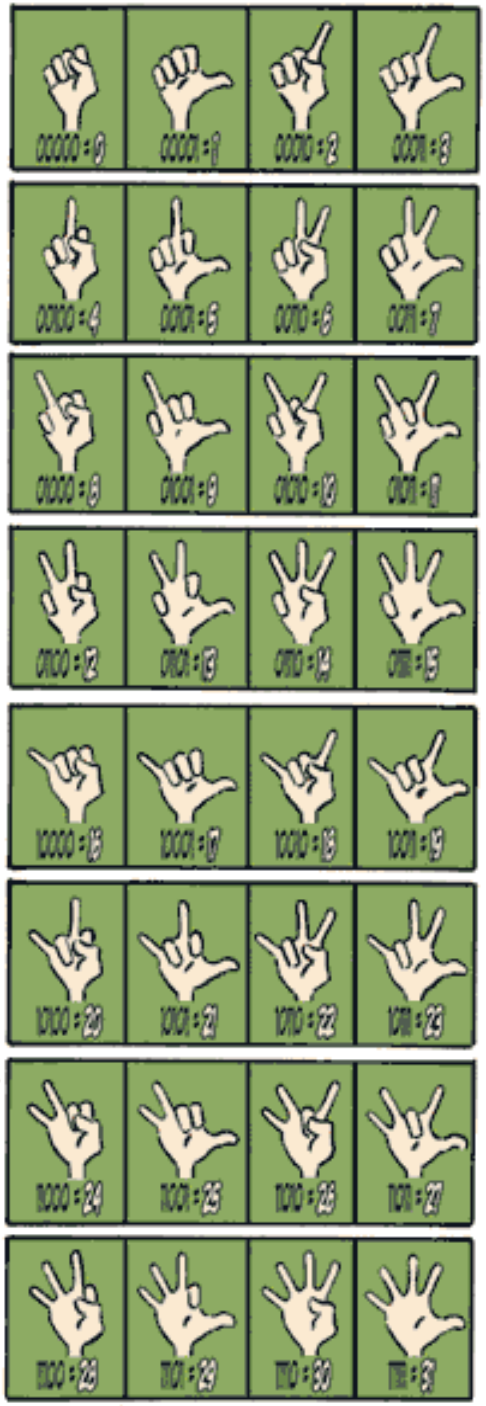
HOWTOONS STYLE!





FIRST IMAGINE THAT EACH FINGER REPRESENTS A NUMBER. STARTING WITH THE THUMB, THAT WILL BE NUMBER 1. YOUR INDEX FINGER WILL BE NUMBER 2, AND YOUR MIDDLE FINGER NUMBER 4.

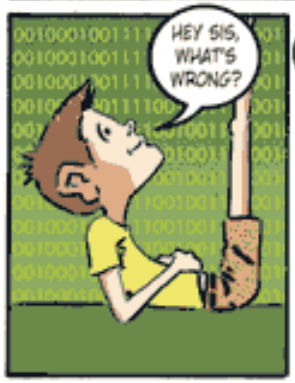
ARE YOU NOTICING A PATTERN HERE? ALL PROCEEDING FINGERS ARE DOUBLE THE ONE BEFORE IT. YOUR NEXT FINGER IS 8, AND ENDS WITH THE PINKY BEING NUMBER 16.



THAT'S PRETTY MUCH IT. ONCE YOU CAN IMAGINE YOUR FINGERS BEING THESE NUMBERS, YOUR READY TO GO. SHOWING CERTAIN FINGERS AND THEN ADDING THEM IS WHAT NUMBER YOU GET!

FOR INSTANCE THE HANG LOOSE SIGN IS:

SEE CHART FOR 0-31.



Gehele getallen

- vaste geheugengrootte
  - er is steeds een grootste en kleinste getal dat kan worden voorgesteld in het geheugen
- interne voorstelling
  - *unsigned* (enkel positieve getallen) binair getal waarvoor vast aantal bits gereserveerd zijn
- voorbeeld: `int` (8 bits)

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	1	0	0	1	1	0	1
128	64	32	16	8	4	2	1

Σ = 77

UNIVERSITEIT GENT

Gehele getallen

- vaste geheugengrootte
  - er is steeds een grootste en kleinste getal dat kan worden voorgesteld in het geheugen
- interne voorstelling
  - *unsigned* (enkel positieve getallen) binair getal waarvoor vast aantal bits gereserveerd zijn
- voorbeeld: `int` (8 bits)

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	0	1	0	1	0	1	0
128	64	32	16	8	4	2	1

Σ = 170

UNIVERSITEIT GENT

Gehele getallen

- vaste geheugengrootte
  - er is steeds een grootste en kleinste getal dat kan worden voorgesteld in het geheugen
- interne voorstelling
  - *unsigned* (enkel positieve getallen) binair getal waarvoor vast aantal bits gereserveerd zijn
- voorbeeld: `int` (8 bits)

maximale waarde

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	1	1	1	1	1	1
128	64	32	16	8	4	2	1

Σ = 255

UNIVERSITEIT GENT

Gehele getallen

- vaste geheugengrootte
  - er is steeds een grootste en kleinste getal dat kan worden voorgesteld in het geheugen
- interne voorstelling
  - *signed* (positieve en negatieve getallen) *twee-complement* binair getal met tekenbit waarvoor vast aantal bits gereserveerd zijn

eerste bit = nul
⇒ positief getal

- voorbeeld: `int` (16 bits)

+/-	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0
Σ	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1

= 1032

Gehele getallen

- vaste geheugengrootte
  - er is steeds een grootste en kleinste getal dat kan worden voorgesteld in het geheugen
- interne voorstelling
  - *signed* (positieve en negatieve getallen) *twee-complement* binair getal met tekenbit waarvoor vast aantal bits gereserveerd zijn

voorbeeld: `int` (16 bits)

twee complement

eerste bit = één
⇒ negatief getal

+/-	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
1	0	0	0	0	1	0	0	0	0	0	0	0	1	1	1
	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
	0	0	0	0	1	0	0	0	0	0	0	0	1	1	1

Gehele getallen

- vaste geheugengrootte
  - er is steeds een grootste en kleinste getal dat kan worden voorgesteld in het geheugen
- interne voorstelling
  - *signed* (positieve en negatieve getallen) *twee-complement* binair getal met tekenbit waarvoor vast aantal bits gereserveerd zijn
- voorbeeld: **`int`** (16 bits)

+/-	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
1	0	0	0	0	1	0	0	0	0	0	0	0	1	1	1
Σ	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1

= -1031 - 1 = -1032

Reële getallen

- vaste geheugengrootte
  - elk gegevenstype voor reële getallen heeft steeds een bepaald aantal bits voor *mantisse* en *exponent*
  - niet alle reële getallen kunnen voorgesteld worden in het computergeheugen (nauwkeurigheid, grootte)
- interne voorstelling
  - als *drijvende-komma getal (floating point number)*
- voorbeeld:

$$6{,}4576 \times 10^{12}$$

mantisse

exponent

UNIVERSITEIT GENT

Reële waarden

- punt als decimaal scheidingsteken
- wetenschappelijke notatie: e

```
# notatie van reële waarden
>>> x = 1.5
>>> y = 1e100          # zelfde als 10**100 (1 googol)
>>> z = .5
>>> x, y, z
(1.5, 1e+100, 0.5)

# verkeerde betekenis
>>> x = 1,5            # tuple
>>> x
(1, 5)
```

UNIVERSITEIT GENT

Vragen of opmerkingen ?

UNIVERSITEIT GENT

The sky is the limit …

Dalai Lama
1935-vandaag

"Open your arms to change,
but don't let go of your values."

— Dalai Lama

UNIVERSITEIT GENT