Universiteit Gent
Faculteit Wetenschappen

Programmeren in Python

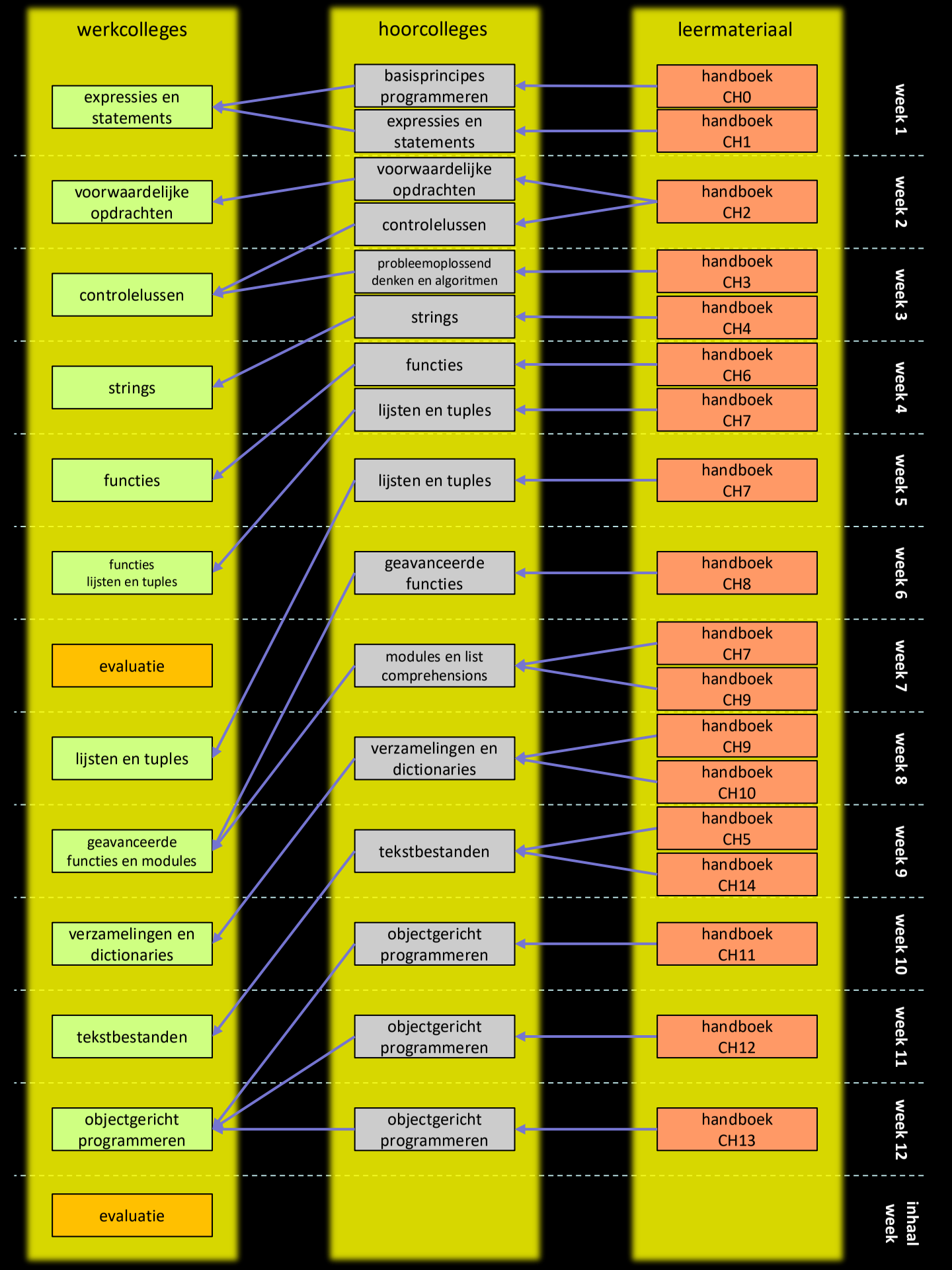
verzamelingen en dictionaries

prof. dr. Peter Dawyndt

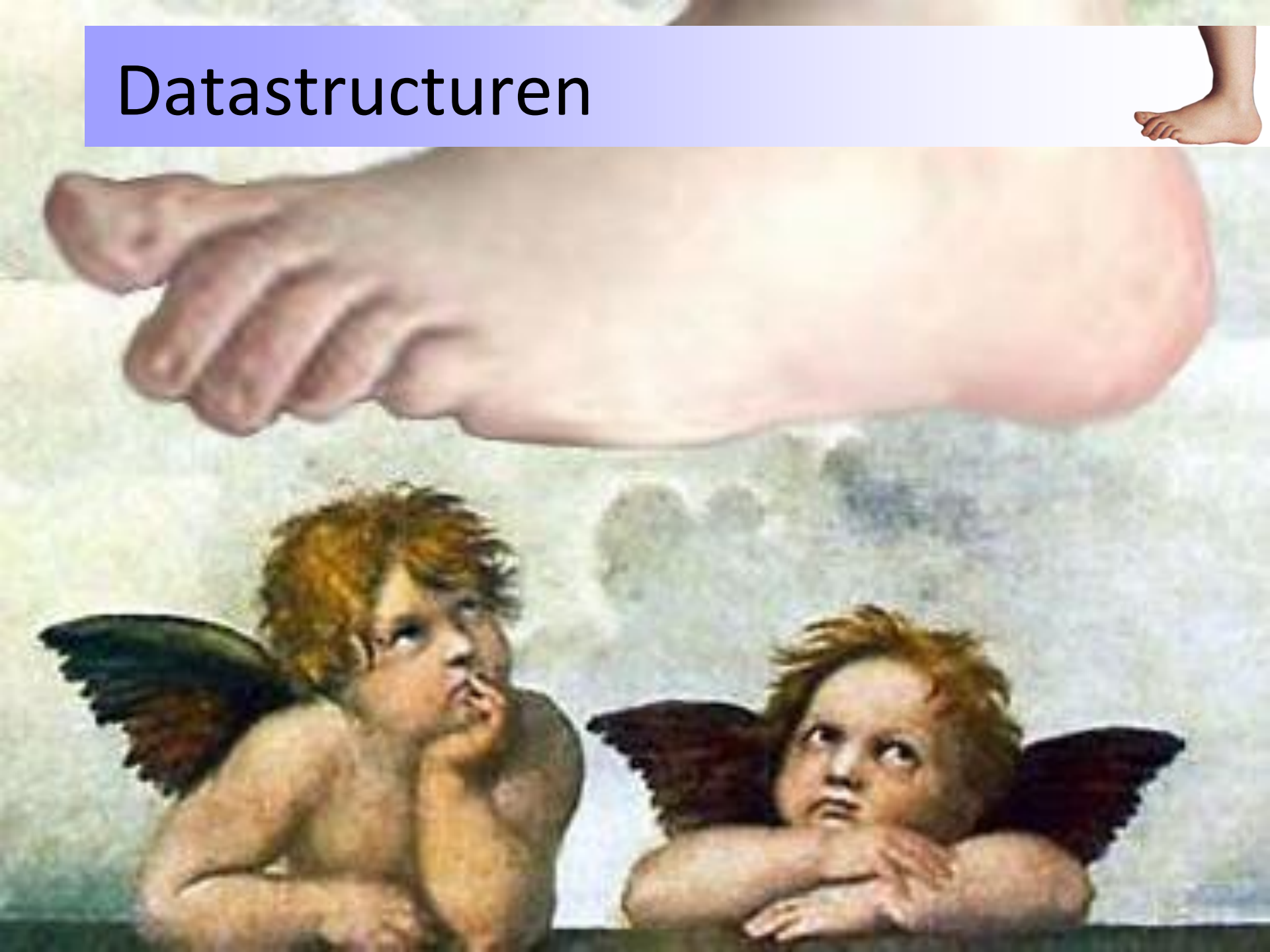
peter.dawyndt@ugent.be

@dawyndt

UNIVERSITEIT GENT

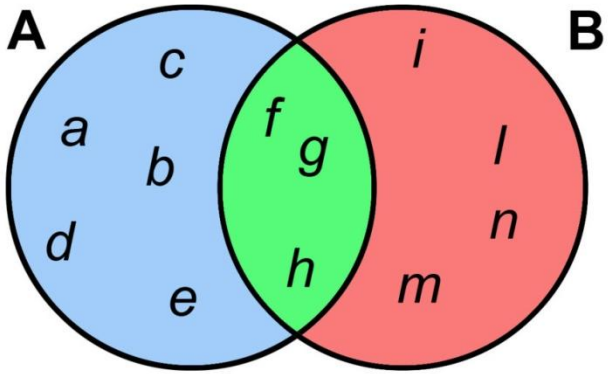

werkcolleges
hoorcolleges
leermateriaal
expressies en statements
voorwaardelijke opdrachten
controlelussen
strings
functies
functies lijsten en tuples
evaluatie
lijsten en tuples
geavanceerde functies en modules
verzamelingen en dictionaries
tekstbestanden
objectgericht programmeren
evaluatie
basisprincipes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

Datastructuren



Datastructuren

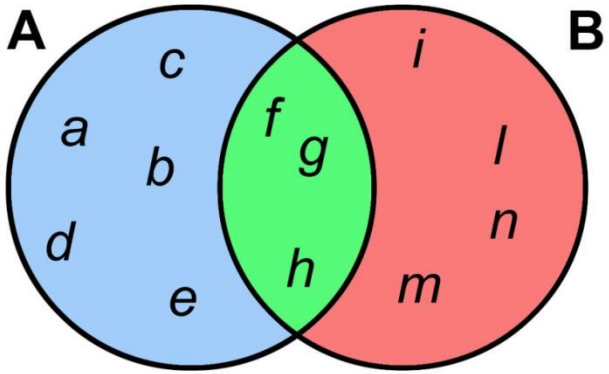
- wereld bestaat niet enkel uit lijsten
  - niet noodzakelijk beste datastructuur voor elk probleem, ondanks feit we deze datastructuur als eerste behandelden
  - in deze les introduceren we nog twee andere datastructuren
    - die toelaten om programma's te schrijven die zowel eenvoudiger als efficiënter zijn
  - we zullen daarbij ook aangeven wat we precies bedoelen met "efficiënter"
- eerste nieuwe datastructuur: verzamelingen



UNIVERSITEIT GENT

Verzamelingen

- **verzameling**: ongeordende collectie verschillende objecten
  - ongeordend: objecten opgezocht via waarde en niet via positie
  - verschillend: zelfde object kan hoogstens één keer voorkomen
- fundamenteel concept in wiskunde
  - ontbreekt in meeste programmeertalen
  - nochtans handig bij probleemoplossend denken



UNIVERSITEIT GENT

Verzamelingen

- **`set`** is ingebouwd gegevenstype van Python
  - verzameling aanmaken met **`set()`** of **`{e1, e2, …}`**
  - gegevens toevoegen, verwijderen, testen op aanwezigheid, …
  - **`set(i)`** zet *iterable object* **`i`** om in verzameling

```
>>> klinkers = set()
>>> for letter in 'aieoeiaoaaeieou':
...     klinkers.add(letter)
...
>>> klinkers
{'a', 'i', 'e', 'u', 'o'}
>>> set('hottentottententententoonstelling')
{'e', 'g', 'i', 'h', 'l', 'o', 'n', 's', 't'}
>>> {x // 2 for x in range(10)}
{0, 1, 2, 3, 4}
```

UNIVERSITEIT GENT

Methoden

- zoals alle objecten hebben ook verzamelingen methoden
  - onderstaande methoden wijzigen verzameling *in place* en geven waarde **None** als resultaat terug

```
>>> laag = {0, 1, 2, 3, 4}
>>> laag.remove(0)              # verwijderen
>>> laag
{1, 2, 3, 4}
>>> laag.add(3)                 # toevoegen
>>> laag.add(9)
>>> laag
{1, 2, 3, 4, 9}
>>> laag.clear()                # leeg maken
>>> laag
set()
```

UNIVERSITEIT GENT

Methoden

- zoals alle objecten hebben ook verzamelingen methoden
  - onderstaande methoden geven nieuwe verzameling terug

```
>>> tien = set(range(10))
>>> laag = {0, 1, 2, 3, 4}
>>> oneven = {1, 3, 5, 7, 9}
>>> laag.difference(oneven)                  # verschil
{0, 2, 4}
>>> laag
{0, 1, 2, 3, 4}
>>> oneven.difference(laag)                  # asymmetrisch
{9, 5, 7}
>>> laag.symmetric_difference(oneven)# symmetrisch
{0, 2, 4, 5, 7, 9}
```

UNIVERSITEIT GENT

Methoden

- zoals alle objecten hebben ook verzamelingen methoden
  - onderstaande methoden geven nieuwe verzameling terug
  - kunnen ook uitgedrukt worden met operator

```
>>> tien = set(range(10))
>>> laag = {0, 1, 2, 3, 4}
>>> oneven = {1, 3, 5, 7, 9}
>>> laag - oneven                    # verschil
{0, 2, 4}
>>> laag
{0, 1, 2, 3, 4}
>>> oneven - laag                    # asymmetrisch
{9, 5, 7}
>>> laag ^ oneven                    # symmetrisch
{0, 2, 4, 5, 7, 9}
```

UNIVERSITEIT GENT

Methoden

- zoals alle objecten hebben ook verzamelingen methoden
  - onderstaande methoden geven nieuwe verzameling terug

```
>>> tien = set(range(10))
>>> laag = {0, 1, 2, 3, 4}
>>> oneven = {1, 3, 5, 7, 9}
>>> laag.intersection(oneven)      # doorsnede
{1, 3}
>>> laag.union(oneven)             # unie
{0, 1, 2, 3, 4, 5, 7, 9}
>>> laag.issubset(tien)            # deelverzameling
True
>>> laag.issuperset(oneven)        # bovenverzameling
False
```

UNIVERSITEIT GENT

Methoden

- zoals alle objecten hebben ook verzamelingen methoden
  - onderstaande methoden geven nieuwe verzameling terug
  - kunnen ook uitgedrukt worden met operator

```
>>> tien = set(range(10))
>>> laag = {0, 1, 2, 3, 4}
>>> oneven = {1, 3, 5, 7, 9}
>>> laag & oneven                    # doorsnede
{1, 3}
>>> laag | oneven                    # unie
{0, 1, 2, 3, 4, 5, 7, 9}
>>> laag <= tien                     # deelverzameling
True
>>> laag >= oneven                   # bovenverzameling
False
```

UNIVERSITEIT GENT

Verzamelingen doorlopen

- verzamelingen zijn *iterable objects*

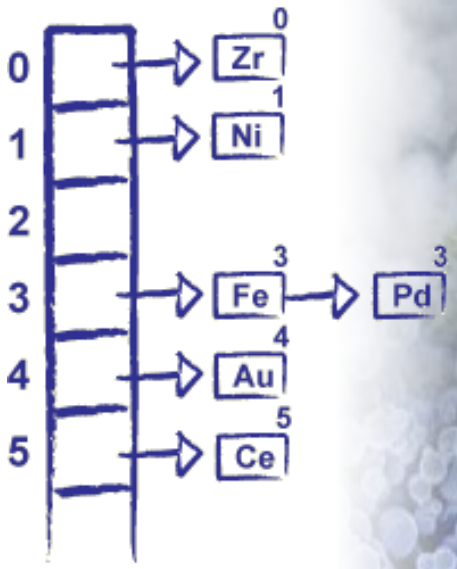
```
>>> monty_python = ['Graham Chapman', 'Eric Idle',
...                 'Terry Gilliam', 'Terry Jones',
...                 'John Cleese', 'Michael Palin']
>>> voornamen = [v for v, f in
...     [naam.split() for naam in monty_python]]
>>> voornamen
['Graham','Eric','Terry','Terry','John','Michael']
>>> unieke_voornamen = set(voornamen)
>>> for naam in unieke_voornamen:
...     print(naam, end=' ')
...
Michael John Graham Eric Terry
```

UNIVERSITEIT GENT

Verzameling als datastructuur

- algemene doelstelling
  - objecten zo snel mogelijk opzoeken
  - objecten toevoegen en verwijderen niet nodeloos traag

- mogelijke oplossing: hashtabel (*hash table*)
  - bereken hashwaarde voor elk object dat moet toegevoegd worden
  - sla object op die positie op in lijst
  - botsingen → waarden gegroepeerd in deellijst
  - goede hashfunctie → weinig botsingen

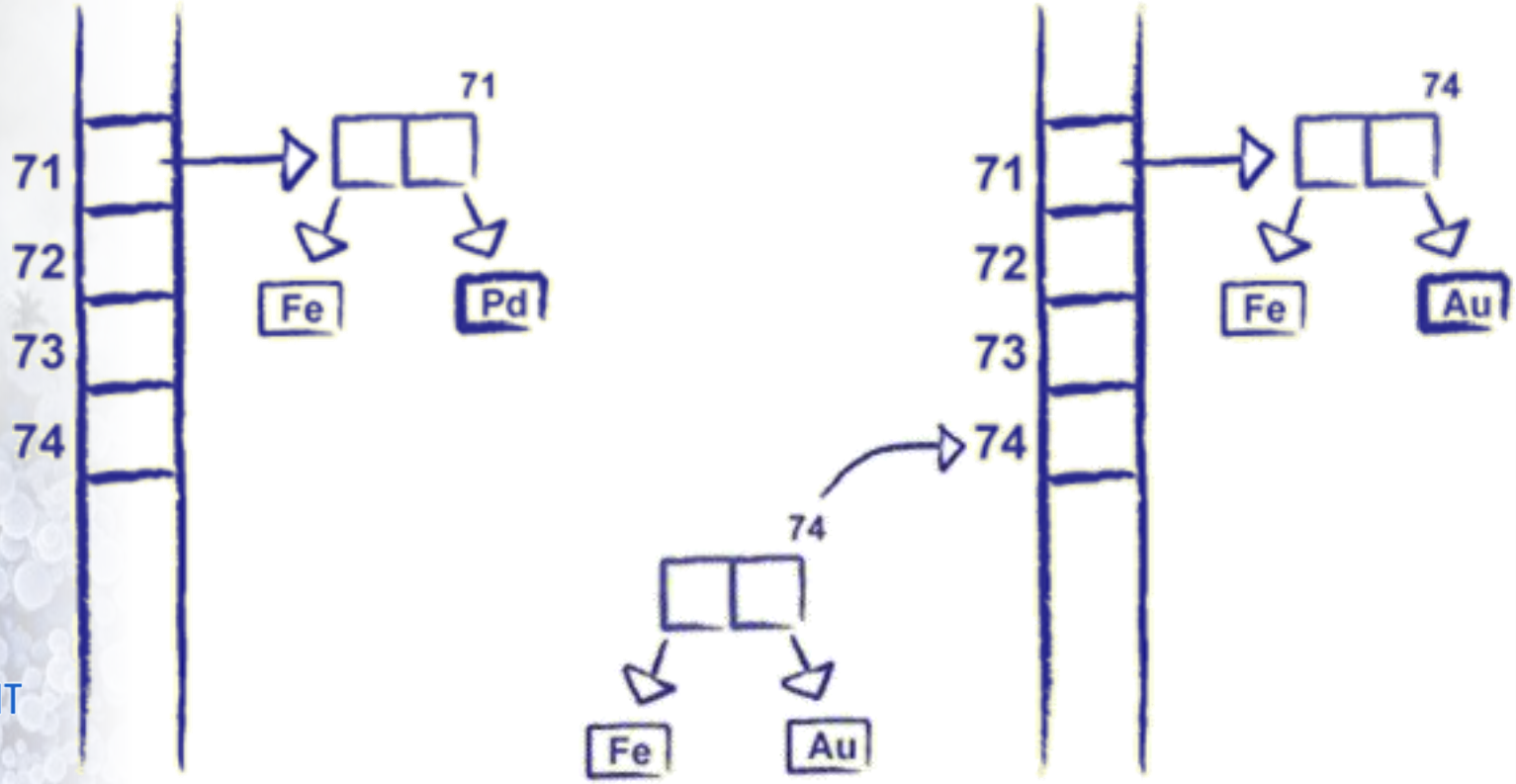


  - resultaat: objecten kunnen in *constante tijd* opgezocht worden, ongeacht hoeveel objecten er in de verzameling zitten

UNIVERSITEIT GENT

Verzameling als datastructuur

- werkt enkel indien hashwaarde nooit wijzigt nadat object werd toegevoegd
  - anders zou object op verkeerde plaats zitten



UNIVERSITEIT GENT

Verzameling als datastructuur

- werkt enkel indien hashwaarde nooit wijzigt nadat object werd toegevoegd
  - anders zou object op verkeerde plaats zitten
  - verzamelingen mogen daarom in Python enkel onveranderlijke (*immutable*) objecten bevatten
    - booleans, getallen, strings, tuples, …
    - **geen** lijsten

```
>>> monty_python = set()
>>> monty_python.add('Graham Chapman')
>>> monty_python
{'Graham Chapman'}
>>> monty_python.add(('Eric', 'Idle'))
>>> monty_python
{'Graham Chapman', ('Eric', 'Idle')}
>>> monty_python.add(['Terry', 'Gilliam'])
TypeError: unhashable type: 'list'
```

UNIVERSITEIT GENT

Verzameling als datastructuur

- werkt enkel indien hashwaarde nooit wijzigt nadat object werd toegevoegd
  - anders zou object op verkeerde plaats zitten
  - verzamelingen mogen daarom in Python enkel onveranderlijke (immutable) objecten bevatten
    - dit is één van de bestaansredenen van tuples
      - hierdoor kunnen objecten met meerdere onderdelen gehashed worden

```
>>> monty_python = set()
>>> monty_python.add('Graham Chapman')
>>> monty_python
{'Graham Chapman'}
>>> monty_python.add(('Eric', 'Idle'))
>>> monty_python
{'Graham Chapman', ('Eric', 'Idle')}
>>> monty_python.add(['Terry', 'Gilliam'])
TypeError: unhashable type: 'list'
```

UNIVERSITEIT GENT

Onveranderlijke verzamelingen

- wat met verzamelingen van verzamelingen?
  - verzamelingen zijn zelf veranderlijk (*mutable*)
  - hierdoor kunnen elementen toegevoegd en verwijderd worden
- Python ondersteunt ook "bevroren" verzamelingen
  - kunnen niet gewijzigd worden na hun creatie
  - worden bijna altijd aangemaakt op basis van container

```
>>> monty_python = set()
>>> terries = frozenset(['Terry Jones', 'Terry Gilliam'])
>>> monty_python.add(terries)
>>> monty_python
set([frozenset(['Terry Gilliam', 'Terry Jones'])])
>>> terries.add('Michael Palin')
AttributeError: 'frozenset' object has no attribute 'add'
```

UNIVERSITEIT GENT

Ontwerp programmeertaal

- sommige programmeertalen (maar Python dus niet) laten veranderlijke elementen toe in verzamelingen
  - gebruikers mogen die niet wijzigen nadat ze zijn toegevoegd
  - oorzaak van moeilijk op te sporen bugs
- in sommige programmeertalen (maar niet in Python) kennen objecten de verzamelingen waartoe ze behoren
  - "verplaats" ze wanneer hun waarden wijzigen
  - oorzaak van trage programma's voor slechts paar gevallen waarin dit strikt noodzakelijk is
- elk programmeertaal moet dit soort ontwerpsoverwegingen maken

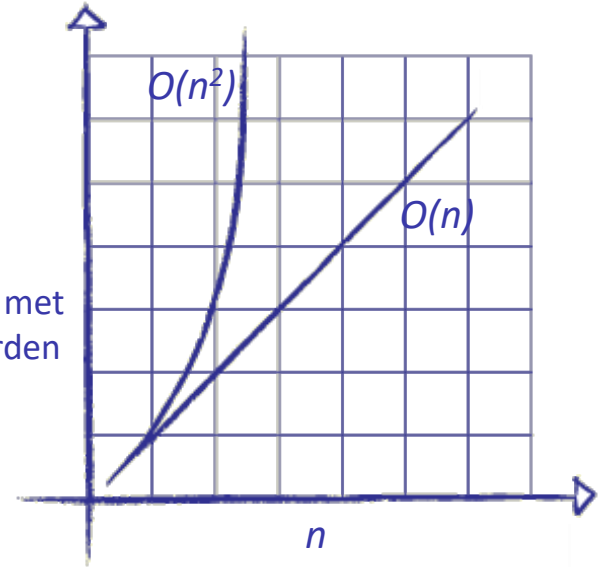
UNIVERSITEIT GENT

Efficiëntie

- loont het de moeite om verzamelingen te gebruiken?
  - datastructuur met *n* unieke woorden opbouwen

```
if woord in woorden
```

  - `type(woorden) == list`
    - elke controle vereist gemiddeld *n*/2 vergelijkingen
    - lijst met *n* unieke woorden opbouwen: *n*(*n*+1)/4 vergelijkingen
  - `type(woorden) == set`
    - set met *n* unieke woorden opbouwen: *n* vergelijkingen



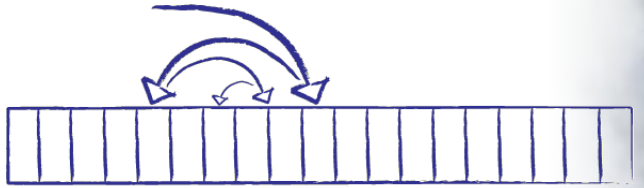
datastructuur met *n* unieke woorden opbouwen

UNIVERSITEIT GENT

Efficiëntie

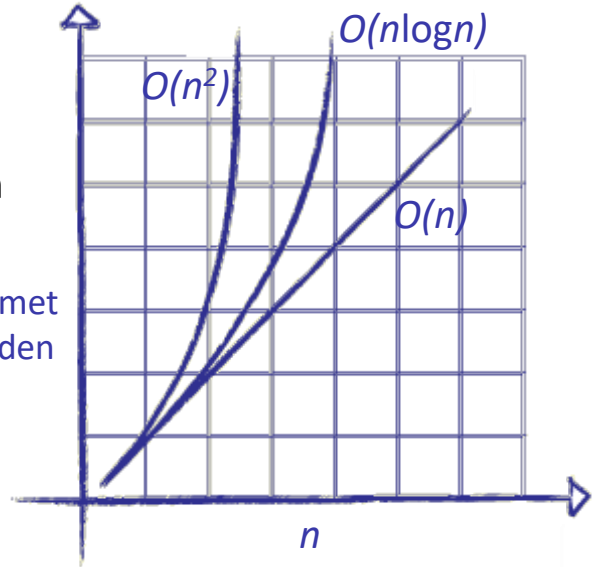
- loont het de moeite om verzamelingen te gebruiken?
  - datastructuur met *n* unieke woorden opbouwen

```
if woord in woorden
```



binair zoeken

  - `type(woorden) == list`
    - werkt sneller als we de lijst gesorteerd houden
    - *k* controles volstaan om waarde te vinden in lijst van lengte 2^k
      - lijst met *n* waarden kan men in $\log_2 n$ stappen doorzoeken
      - lijst met *n* unieke woorden opbouwen: $n\log_2 n$ vergelijkingen



datastructuur met *n* unieke woorden opbouwen

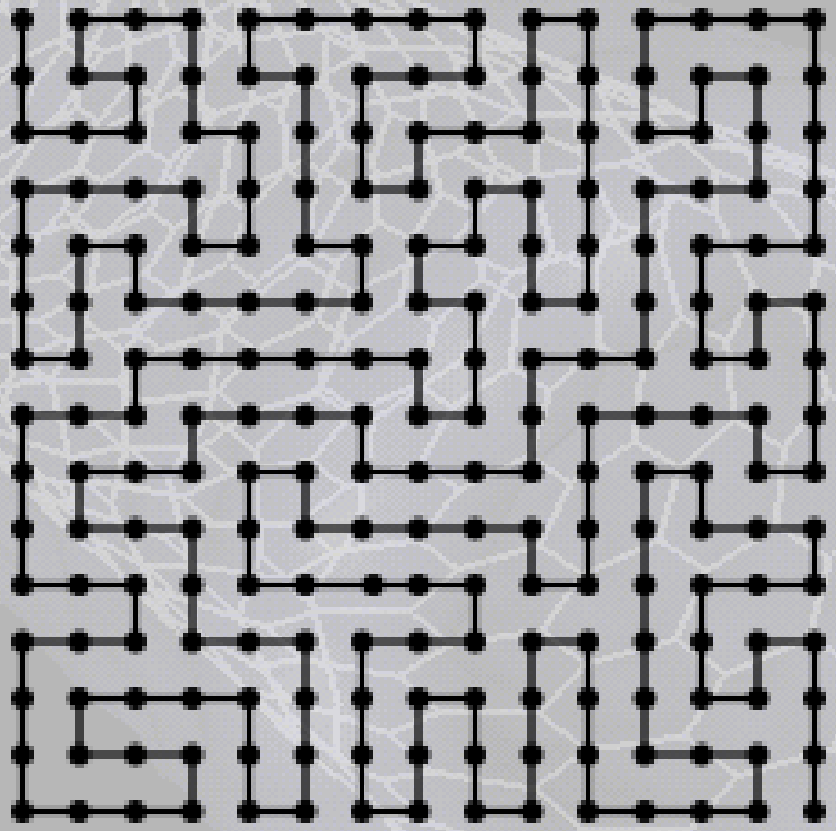
UNIVERSITEIT GENT

Complexiteit

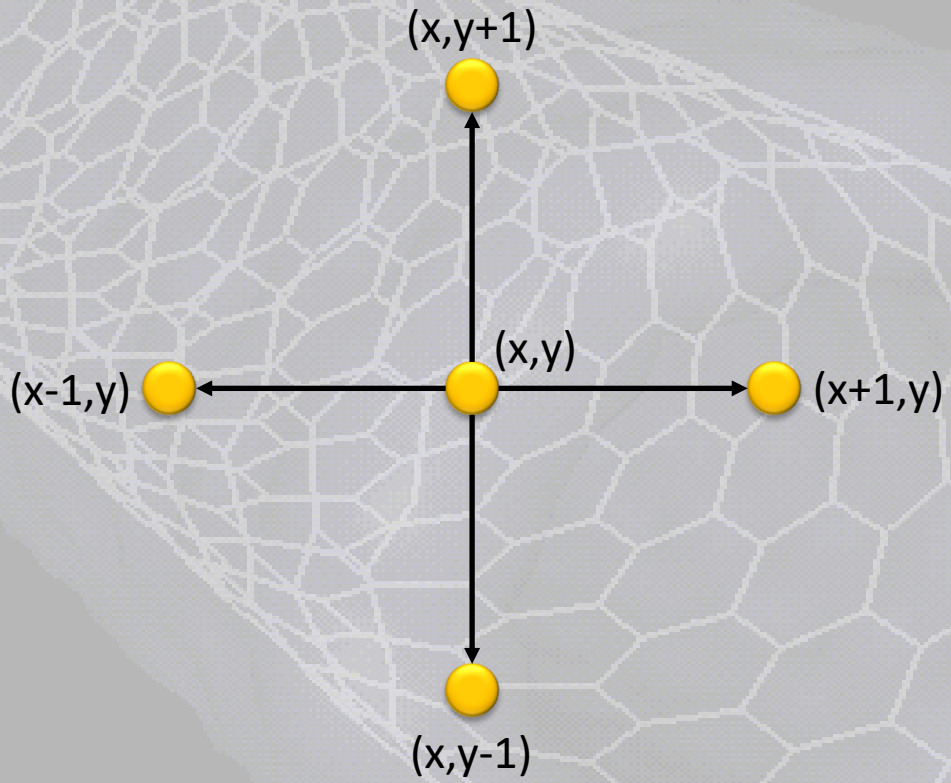
- algoritmische complexiteit
  - relatie tussen grootte van probleem en uitvoeringstijd
  - meestal uitgedrukt in functie van bovengrenzen
  - $f(x) = O(g(x))$: als $f(x) < kg(x)$ voor grote x en constante k
- voorbeeld
  - $O(1)$ iets neemt evenveel tijd in beslag los van de grootte van het probleem
  - $O(\log n)$ uitvoeringstijd groeit logaritmisch met de grootte van het probleem
  - $O(n)$ uitvoeringstijd groeit in verhouding tot de grootte van het probleem
- lijst met unieke woorden opbouwen is $O(n^2)$
  - $n(n+1)/4 = n^2/4 + n/4 \quad \sim \quad n^2$ als n groot wordt

UNIVERSITEIT GENT

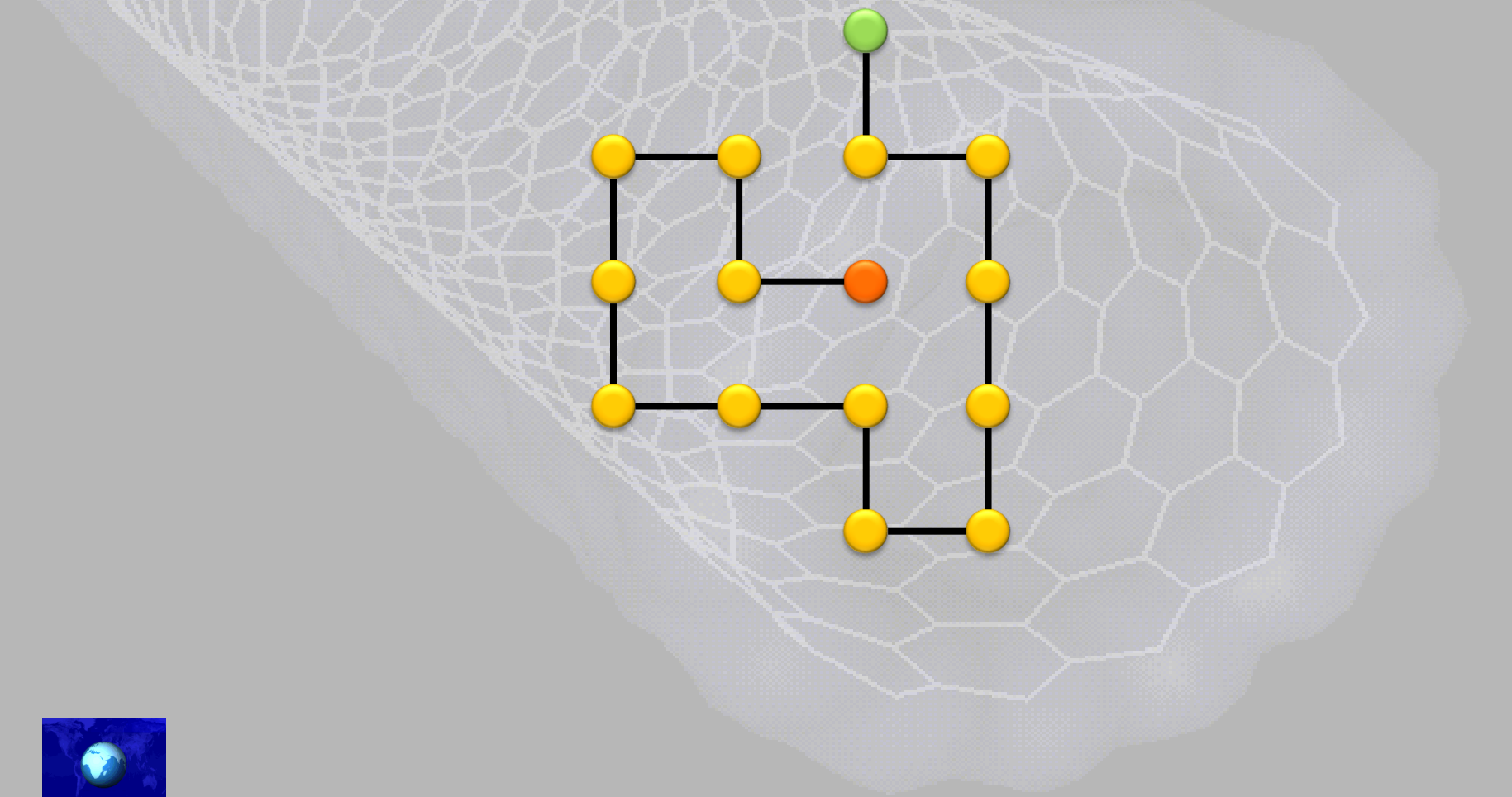
Polymeren simuleren



Polymeren simuleren

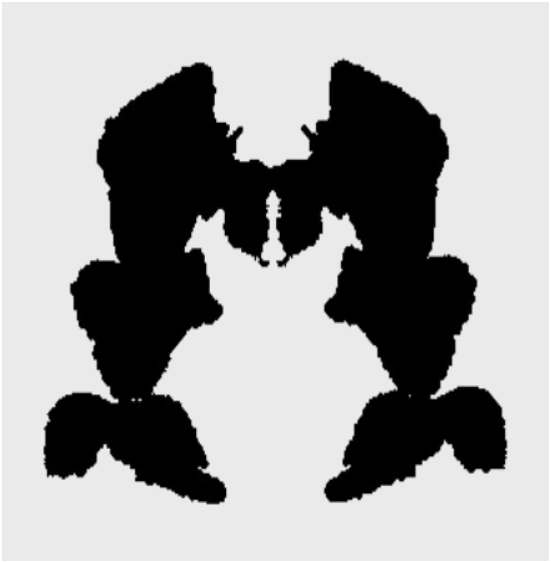


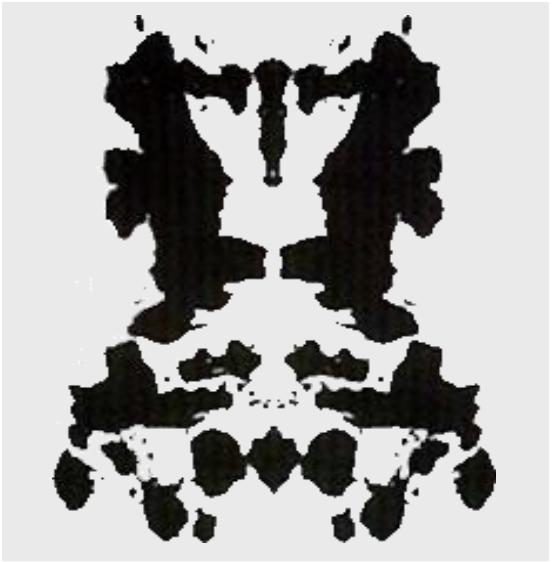
Polymeren simuleren



Dictionaries

Menselijke associatiegedrag





UNIVERSITEIT GENT

Menselijke associatiegedrag

APPEL

Waarmee associeer je dit woord ?

UNIVERSITEIT GENT

Heb je hieraan gedacht …

lekker

fruit

GROEN

UNIVERSITEIT GENT

# of eerder hieraan ?

UNIVERSITEIT GENT

Logo's na financiële crisis

DOWNJONES

NOKIA
Disconnecting People

PLEASE BUY

2M

Will work for food

DEFAULT

citigroup

LG
Life's tough

CRISISLER

DELL

YAHOO?

BAD YEAR

UNIVERSITEIT GENT

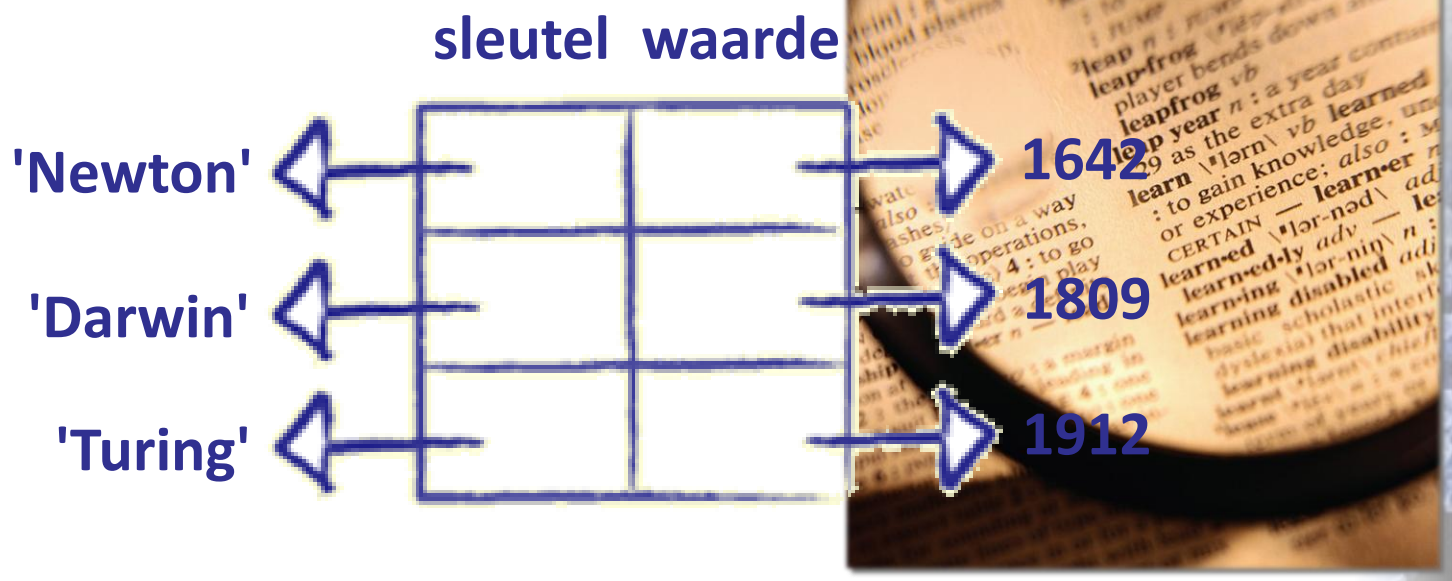
FIASCO SYSTEMS

Fail

Ferrari

Dictionaries

- **rationale**: gegevens opslaan bij elementen uit verzameling
- **dictionary**: ongeordende veranderlijke afbeelding
  - geen collectie maar *afbeelding* (*mapping*)
  - associeert objecten met elk van zijn sleutels
  - alternatieve benamingen: map, hash, associatieve array
  - vaak voorgesteld als tabel met twee kolommen



UNIVERSITEIT GENT

Aanmaken en indexeren

- dictionary aanmaken: sleutel/waarde paren tussen accolades
  - **{ }** stelt lege dictionary voor
- waarde opzoeken: sleutel als index tussen vierkante haken
  - enkel bestaande sleutels zijn toegankelijk
  - net zoals je enkel bestaande elementen van lijst kunt indexeren

```
>>> geboorte = {'Newton':1642, 'Darwin':1809}
>>> f"Darwin's geboortejaar: {geboorte['Darwin']}"
"Darwin's geboortejaar: 1809"
>>> f"Newton's geboortejaar: {geboorte['Newton']}"
"Newton's geboortejaar: 1642"
>>> f"Turing's geboortejaar: {geboorte['Turing']}"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'Turing'
```

UNIVERSITEIT GENT

Bijwerken

- object koppelen aan sleutel
  - sleutel komt nog niet voor: nieuw paar aanmaken
  - sleutel komt reeds voor: waarde gekoppeld aan sleutel overschrijven
- paar verwijderen: **`del d[`*`sleutel`*`]`**
  - enkel paar met bestaande sleutel kan verwijderd worden

```
>>> geboorte = {}
>>> geboorte['Darwin'] = 1809
>>> geboorte['Newton'] = 1942      # oeps
>>> geboorte['Newton'] = 1642
>>> geboorte['Turing'] = 1912
>>> geboorte
{'Turing': 1912, 'Darwin': 1809, 'Newton': 1642}
>>> del geboorte['Turing']
>>> geboorte
{'Darwin': 1809, 'Newton': 1642}
>>> del geboorte['Faraday']
KeyError: 'Faraday'
```

UNIVERSITEIT GENT

Lidmaatschap

- **`s in d`**
  - test of er een paar met sleutel **`s`** voorkomt in dictionary **`d`**
  - inconsistent met lijsten, waar getest wordt of een waarde in de lijst voorkomt, niet een gegeven index

```
>>> geboorte = {'Newton':1642, 'Darwin':1809}
>>> for naam in ['Newton', 'Turing']:
...     if naam in geboorte:
...         print(naam, geboorte[naam])
...     else:
...         print(f'Wie is {naam} ?')
...
Newton 1642
Wie is Turing ?
```

UNIVERSITEIT GENT

Controlelussen

- **`for s in d`**
  - overloopt sleutels van dictionary **`d`** (niet de waarden)
  - verschilt dus ook met lijsten, waar **`for`** lus de waarden doorloopt en niet de indices

```
>>> geboorte = {
...     'Newton' : 1642,
...     'Darwin' : 1809,
...     'Turing' : 1912
... }
>>> for naam in geboorte:
...     print(naam, geboorte[naam])
...
Turing 1912
Newton 1642
Darwin 1809
```

UNIVERSITEIT GENT

Dictionarymethoden

- inderdaad, dictionaries zijn ook objecten …

```
>>> geboorte = {'Newton':1642,'Darwin':1809,'Turing':1912}
```

methode	functionaliteit	voorbeeld	resultaat
`clear`	dictionary leegmaken	`d.clear()`	geeft `None` terug `d` is nu lege dictionary
`get`	geeft object terug dat met gegeven sleutel geassocieerd is, of standaard-waarde indien sleutel niet bestaat	`d.get('x', 99)`	geeft `d['x']` terug als `'x' in d`, of anders de waarde `99`
`keys`	geeft een iterator van sleutels van de dictionary terug; elementen zijn dus gegarandeerd uniek	`geboorte.keys()`	`['Turing', 'Newton', 'Darwin']`
`values`	geeft een iterator van waarden van de dictionary terug; waarden zijn niet noodzakelijk uniek	`geboorte.values()`	`[1912, 1642, 1809]`

`items`	geeft een iterator van (sleutel, waarde)-paren terug als iterable van tuples	`geboorte.items()`	`[('Turing', 1912),`
			`('Newton', 1642),`
			`('Darwin', 1809)]`

`update`	kopieert sleutels en waarden van de ene dictionary naar een andere	`d2 = {}` `d2.update(d)`	

Dictionarymethoden

```
>>> geboorte = {
...     'Newton': 1642,
...     'Darwin': 1809,
...     'Turing': 1912
... }
>>> f'sleutels: {list(geboorte.keys())}'
sleutels: ['Turing', 'Newton', 'Darwin']
>>> f'waarden: {list(geboorte.values())}'
waarden: [1912, 1642, 1809]
>>> f'items: {list(geboorte.items())}'
items: [('Turing', 1912), ('Newton', 1642), ('Darwin', 1809)]
>>> f'get: {geboorte.get('Curie', 1867)}'
get: 1867
>>> temp = {'Curie': 1867, 'Hopper': 1906, 'Franklin': 1920}
>>> geboorte.update(temp)
>>> geboorte
{'Curie': 1867, 'Darwin': 1809, 'Franklin': 1920,
 'Turing': 1912, 'Newton': 1642, 'Hopper': 1906}
>>> temp.clear()
>>> temp
{}
```

UNIVERSITEIT GENT

Cryptogrammen

C	h		l	d							o	w		
Q	m	v	r	b	w	l	f			x	w	k	d	

f	a	s			r			k						
i	o	p	z	l	w		v	f			z	m	l	

	p		i		g	t			k	e	.			
p	c	w	v	f	x	z	v	y	l	.				

Frequentietabel opbouwen

observaties1.py

```
# vogels die moeten geteld worden
vogels = ['stern', 'gans', 'gans', 'havik',
          'stern', 'gans', 'stern']

# dictionary met tellingen opbouwen
observaties = {}
for vogel in vogels:
    if vogel in observaties:
        # reeds gezien, dus verhoog met 1
        observaties[vogel] += 1
    else:
        # nog niet gezien, dus toevoegen aan dictionary
        observaties[vogel] = 1

# dictionary uitschrijven
print(observaties)
```

```
{'havik': 1, 'gans': 3, 'stern': 3}
```

UNIVERSITEIT GENT

Frequentietabel opbouwen

observaties2.py

```
# vogels die moeten geteld worden
vogels = ['stern', 'gans', 'gans', 'havik',
          'stern', 'gans', 'stern']

# dictionary met tellingen opbouwen
observaties = {}
for vogel in vogels:
    # dict.get: aantal tellingen van de vogel ophalen
    # met 0 als standaardwaarde
    observaties[vogel] = observaties.get(vogel, 0) + 1



# dictionary uitschrijven
print(observaties)
```

```
{'havik': 1, 'gans': 3, 'stern': 3}
```

UNIVERSITEIT GENT

Geordend uitschrijven

- sleutels van dictionary zijn niet geordend (~ verzamelingen)
  - willekeurige orde bepaald door (interne) hashfunctie
  - tellingen in alfabetische orde uitschrijven
    - lijst van sleutels opvragen
    - lijst sorteren
    - lijst doorlopen

```
# dictionary geordend uitschrijven
vogels = list(observaties)
vogels.sort()
for vogel in vogels:
    print(vogel, observaties[vogel])
```

```
gans 3
havik 1
stern 3
```

UNIVERSITEIT GENT

Geordend uitschrijven

- sleutels van dictionary zijn niet geordend (~ verzamelingen)
  - willekeurige orde bepaald door (interne) hashfunctie
  - tellingen in alfabetische orde uitschrijven
    - lijst van sleutels opvragen
    - lijst sorteren
    - lijst doorlopen

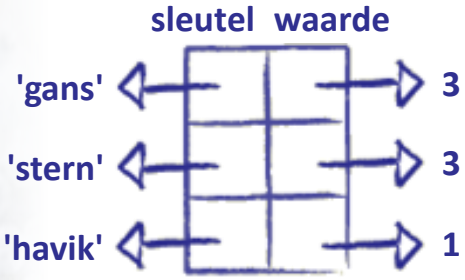
```
# dictionary geordend uitschrijven


for vogel in sorted(observaties):
    print(vogel, observaties[vogel])
```

```
gans 3
havik 1
stern 3
```

UNIVERSITEIT GENT

Dictionary omkeren

- hoe drukken we af volgens aantal tellingen ?
  - dictionary moet omgekeerd worden: sleutels en waarden omkeren

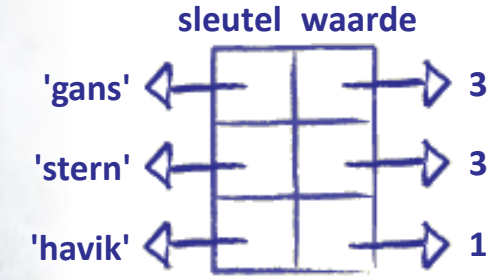


observaties

UNIVERSITEIT GENT

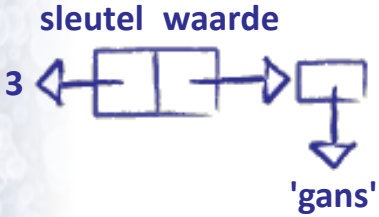
Dictionary omkeren

- hoe drukken we af volgens aantal tellingen ?
  - dictionary moet omgekeerd worden: sleutels en waarden omkeren
  - er kunnen botsingen optreden: waarden zijn niet noodzakelijk uniek
    - wat is het omgekeerde van `{'a':1, 'b':1, 'c':1}` ?



observaties

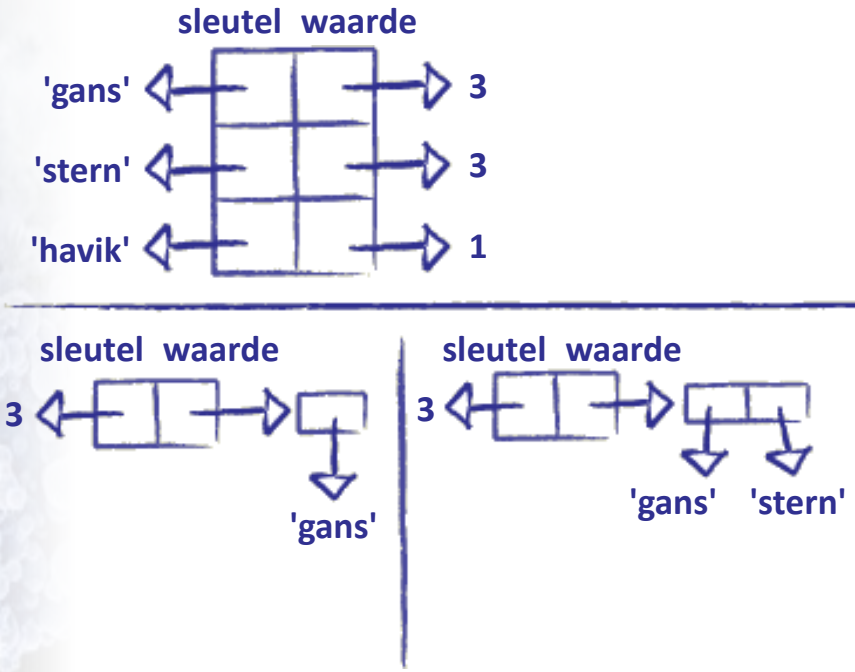
omgekeerd



UNIVERSITEIT GENT

Dictionary omkeren

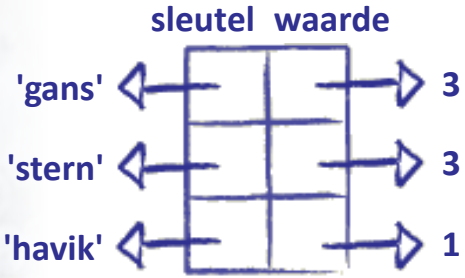
- hoe drukken we af volgens aantal tellingen ?
  - dictionary moet omgekeerd worden: sleutels en waarden omkeren
  - er kunnen botsingen optreden: waarden zijn niet noodzakelijk uniek
    - wat is het omgekeerde van `{'a':1, 'b':1, 'c':1}` ?
  - oplossing: lijst van waarden opslaan ipv één enkele waarde
    - gebruik `dict.get(sleutel, [])` ipv `dict.get(sleutel, 0)`



UNIVERSITEIT GENT

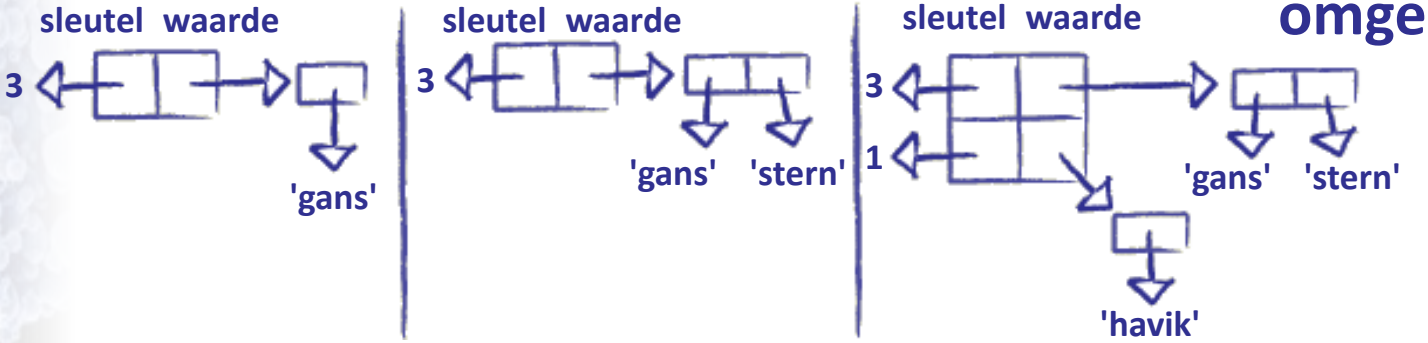
Dictionary omkeren

- hoe drukken we af volgens aantal tellingen ?
  - dictionary moet omgekeerd worden: sleutels en waarden omkeren
  - er kunnen botsingen optreden: waarden zijn niet noodzakelijk uniek
    - wat is het omgekeerde van `{'a':1, 'b':1, 'c':1}` ?
  - oplossing: lijst van waarden opslaan ipv één enkele waarde
    - gebruik `dict.get(sleutel, [])` ipv `dict.get(sleutel, 0)`



observaties

omgekeerd



UNIVERSITEIT GENT

Dictionary omkeren

- hoe drukken we af volgens aantal tellingen ?
  - dictionary moet omgekeerd worden: sleutels en waarden omkeren

```
# dictionary omkeren
omgekeerd = {}
for sleutel, waarde in observaties.items():
    vogels = omgekeerd.get(waarde, [])
    vogels.append(sleutel)
    omgekeerd[waarde] = vogels

# dictionary geordend uitschrijven
for aantal in sorted(omgekeerd):
    print(aantal, omgekeerd[aantal])
```

```
1 ['havik']
3 ['gans', 'stern']
```

UNIVERSITEIT GENT

Dictionary omkeren

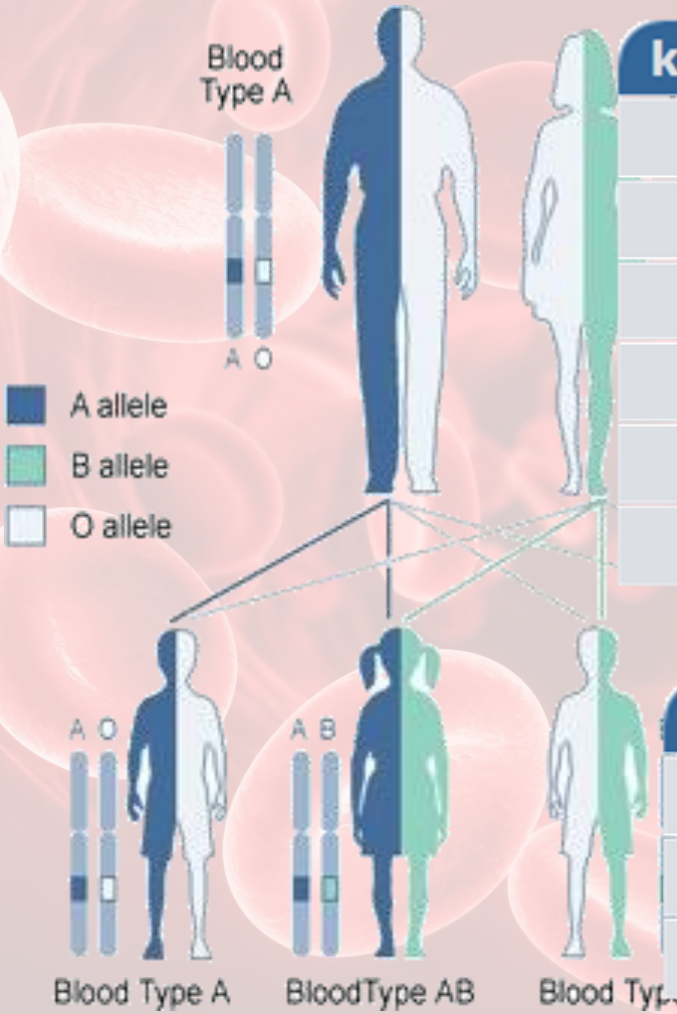
- hoe drukken we af volgens aantal tellingen ?
  - dictionary moet omgekeerd worden: sleutels en waarden omkeren

```
# dictionary omkeren
omgekeerd = {}
for sleutel, waarde in observaties.items():
    if waarde not in omgekeerd:
        omgekeerd[waarde] = []
    omgekeerd[waarde].append(sleutel)

# dictionary geordend uitschrijven
for aantal in sorted(omgekeerd):
    print(aantal, omgekeerd[aantal])
```

```
1 ['havik']
3 ['gans', 'stern']
```

UNIVERSITEIT GENT

Bloedgroepen



kruising	ABO bloedgroep
A en A	A
A en B	AB
A en O	A
B en B	B
B en O	B
O en O	O

kruising	rhesusfactor
+ en +	+
+ en –	+
– en –	–

Benoemde format-argumenten

- complexe opmaak van strings soms moeilijk te lezen
  - vooral als zelfde waarde verschillende keren moet gebruikt worden
  - stringmethode **`format`** kan ook benoemde argumenten meekrijgen
    - gebruik **`{sleutel}`** binnen opmaakspecificatie om aan te geven welk benoemd argument op deze positie moet geplaatst worden

```
geboorte = {'Newton':1642, 'Darwin':1809, 'Turing':1912}
opmaak = '{naam}: {jaar}'
for naam, jaar in geboorte.items():
    print(opmaak.format(naam=naam, jaar=jaar))
```

```
Turing: 1912
Newton: 1642
Darwin: 1809
```

UNIVERSITEIT GENT

Extra benoemde argumenten

- de ** voor **keywords** betekent
  - plaats extra benoemde argumenten in dictionary **keywords**
- ~~laat functies met willekeurig aantal argumenten toe~~

```
-- Do you have any Limburger ?
-- I'm sorry, we're all out of Limburger
client : John Cleese
shopkeeper : Michael Palin
sketch : Cheese Shop Sketch
```

```
def cheeseshop(cheese, **keywords):
    print("-- Do you have any {} ?".format(cheese))
    print("-- I'm sorry, we're all out of {}".format(cheese))
    print("-" * 40)
    for key in sorted(keywords):
        print('{}: {}'.format(key, keywords[key]))

cheeseshop(
    'Limburger',
    client='John Cleese',
    shopkeeper='Michael Palin',
    skecth='Cheese Shop Sketch'
)
```

UNIVERSITEIT GENT

Extra positionele argumenten

- de * voor **arguments** betekent
  - plaats extra positionele argumenten in tuple **arguments**
- er kan slechts één * voorkomen per functie

```
-- Do you have any Limburger ?
-- I'm sorry, we're all out of Limburger
It's very runny, sir.
It's really very, VERY runny, sir.
----------print(argument)----------
client : John Cleese
shopkeeper : Michael Palin
sketch : Cheese Shop Sketch
```

```
cheeseshop(
    'Limburger',
    "It's very runny, sir.",
    "It's really very, VERY runny, sir.",
    client='John Cleese',
    shopkeeper='Michael Palin',
    skecth='Cheese Shop Sketch'
)
```

UNIVERSITEIT GENT

Huiswerk (volgende les)

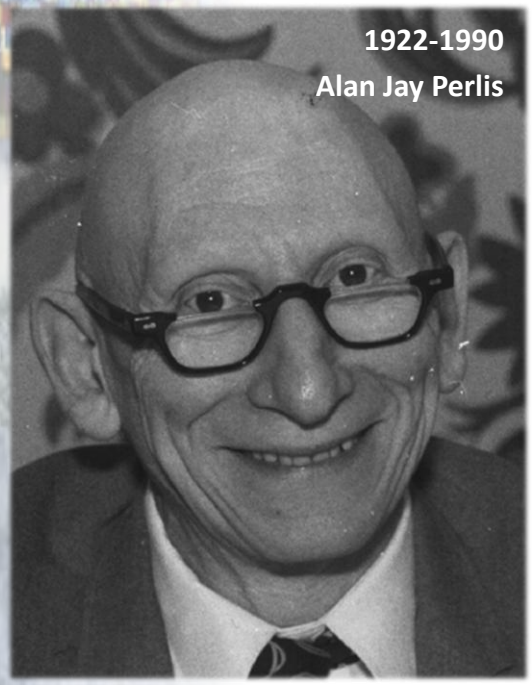
- handboek
  - lees hoofdstuk 5 (files and exceptions)
  - lees hoofdstuk 14 (files and exceptions II)
- voorbeeldopgaven volgende les (reeks 9)
  - Chemische woorden
  - Lithologische kaart

UNIVI
GENT

Vragen of opmerkingen?

UNIVERSITEIT GENT

The sky is the limit…



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

— Alan J. Perlis

UNIVERSITEIT GENT