

Man Chain illustration by [Frits Ahlefeldt-Laurvig](#) | CC BY-ND

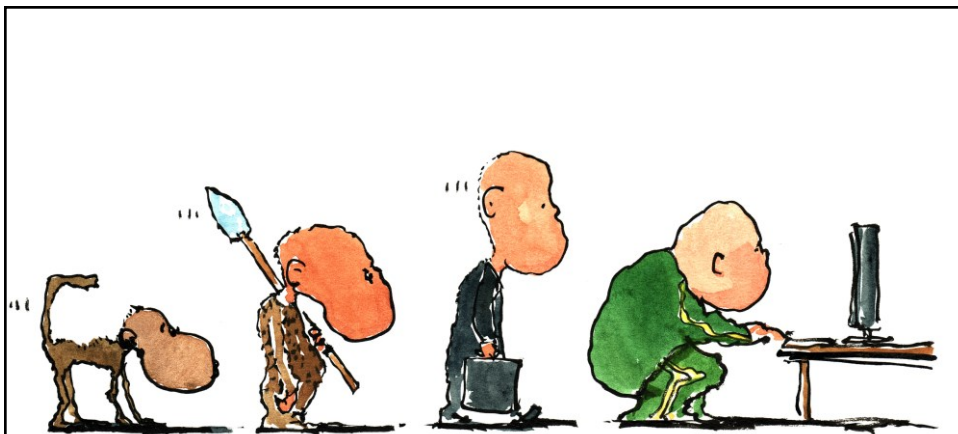
**Week 6: OO, Version Control, Algoritmes**

Programming (TICT-V1PROG-15)  
HBO-ICT propedeuse periode 1 2015-2016

**HOGESCHOOL  
UTRECHT**

## Lesvoorbereiding

- Voorbereiding
  - Student *bestudeert*:
    - Perkovic: Chapter 8 OF NOG BETER:
    - Swaroop (Byte of Python): p. 85 t/m 95
  - Student *maakt*:
    - De opdrachten voor Git en Github



[Man Chain illustration by Frits Ahlefeldt-Laurvig | CC BY-ND](#)

## Namespace / OO / Version Control

Programming (TICT-V1PROG-15)  
HBO-ICT propedeuse periode 1 2015-2016

**HU** HOGESCHOOL  
UTRECHT

### Feed up: Wat gaan we deze les leren?

- Namespace
  - Gebruik van modules
  - Locals
  - Globals
- Object Oriented Programming (OO)
  - Van procedureel naar OO
- Version Control System (Git)
  - Demo
  - Zelfstudie

## Namespaces: Code reuse

- Code die vaker wordt aangeroepen wordt gepackaged in een functie. Dit maakt je code *herbruikbaar*:

```
def do_something_useful(b):
    a = 6
    return a*b

print( do_something_useful(4) )
```

- Maar wat als de developer van de functie *jouw* variabelen gebruikt?

```
def do_something_useful(b):
    a = 6
    return a*b

a = 8
print( do_something_useful(a) )
print(a)
```

## Namespaces: Local / Global

- "*Modularity*" en "*Encapsulation*" dankzij "*namespace*".
- Locale en Globale variabelen.
- Demo op: <http://pythontutor.com/visualize.html>

```
# Print locals and globals
a=3 # Global namespace
z=4 # Global namespace
print()

def doe_bijna_niets():
    z=5 # :ocal namespace
    print("LOCALS: ",locals())
    print("GLOBALS:",globals())

doe_bijna_niets()
```



## Namespaces

- Wijzigen van globals binnen een functie is geen best practice, maar kan wel:

```
a=23
print("A:",a)

def doe():
    global a
    a=88
    print("A binnen functie:",a)

doe()
print("A buiten functie:",a)
```

- Hoe moet dit dan wel?



## Modules als Namespaces

- Als we een module importeren dan is deze module ook een namespace. Het bevat de namen uit de scope van deze module:

```
# importeer de module
import math

# een naam uit de namespace van deze module
print(math.pi)

# die kan je niet verwarren met je eigen variabele pi
pi=3.13
print(pi)

# alle namen
print(dir(math))
```



## Verschillende manieren van importeren

- Welke manier is het beste? "it depends..."

```
# importeer de module
import math
print(math.pi)

# importeer een deel van de module naar het hoofdprogramma.
from math import pi,e
print(pi)

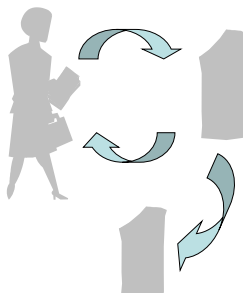
# importeer alle namen (attributen) met de wildcard: *
from math import *
print(pi)
```



## Object Oriented Programming

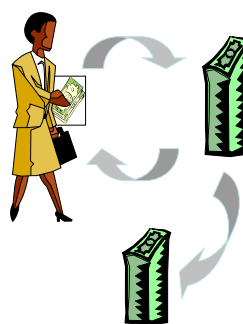
Procedureel vs. Object-georiënteerd

Procedureel



Opname, storting, overboeking

Object-georiënteerd



Klant, geld, rekening



## Object Oriented Programming

- Procedureel:
  - Focus op procedures
  - Alle data wordt gedeeld: geen bescherming
  - Lastig om te veranderen
  - Moeilijk om complexe programma's te bouwen



## Object Oriented Programming

- Een object heeft: eigenschappen (fields) en functies (methods).
- Fields en Methods noemen we "Attributes".



← eigenschappen →  
← methoden →



```

### Object Oriented ###
class BankAccount:
    def __init__(self, name):
        self.balance = 0
        self.name = name

    def withdraw(self, amount):
        self.balance -= amount
        if self.balance < 0:
            print("Saldo te laag")
        return

a = BankAccount("Jan")
a.withdraw(100)
print(a.name, ': ', a.balance)

```

```

### Procedureel ###
a=['Jan',0]
a[1] -= 100

if a[1] < 0:
    print("Saldo te laag")
print(a[0], ': ', a[1])

```

## Objects are everywhere!

```

1
2 # Gebruik klassen van anderen
3 import http.client
4
5 # Maak een nieuw object
6 conn = http.client.HTTPConnection('www.hu.nl')
7
8 # Gebruik een methode
9 conn.request('GET', '/')
10
11 # Gebruik nog een methode
12 print(conn.getresponse().read())
13 print()
14
15 # Lees de waarde van een veld
16 print("HOST", conn.host)
17 print("PORT", conn.default_port)
18
19 # Sluit de connectie met een methode
20 conn.close()
21
22

```

Veld

Methode

## Verschil tussen ene Klasse en een Object

- Klassen geven concepten weer
- Objecten komen overeen met instanties van die concepten.

*class* →



*object*



## Klasse versus Instantie

### ▪ Klasse

- Zichtbaar in broncode
- De code is niet gedupliceerd
- *“De bouwtekening”*

### ▪ Instantie

- *“Een trein gebouwd volgens de tekening.”*

### ▪ Objects

- Eigen kopie van data
- Actief in draaiend programma
- Neemt geheugen in beslag
- Beschikt over opdrachten zoals gedefinieerd in de klasse
- *“De treinen gebouwd volgens deze tekening”*





## Voordelen:

- Code Reuse
  - By packaging code into a class, code can be more easily shared
- Abstraction
  - Details of the implementation can be changed without affecting client code
- Simplicity
  - Simplify complex tasks by creating a simpler interface



## Klasse en Object variabelen:

Twee soorten velden

- Class variabelen: worden gedeeld door alle instanties van een klasse
- Object variabelen: worden niet gedeeld en zijn alleen eigendom van het object zelf, elk object heeft zijn eigen kopie!

```

1 import datetime
2
3 class Person():
4     """Een persoon"""
5
6     unique_number=0
7
8     def __init__(self, fullname, city, birth_year=1900):
9         self.fullname      = fullname
10        self.city           = city
11        self.birth_year     = birth_year
12        self.number         = Person.unique_number
13        Person.unique_number += 1
14
15    def calculate_age(self):
16        return datetime.date.today().year - self.birth_year
17
18    def __str__(self):
19        return("Naam: %s\n" % (self.fullname))
20
21 p1=Person("Jan Janssen","Utrecht")
22 p2=Person("Matthijs Haaksma","Amsterdam",1920)
23 print(p1.birth_year)
24 print(p2)

```

```

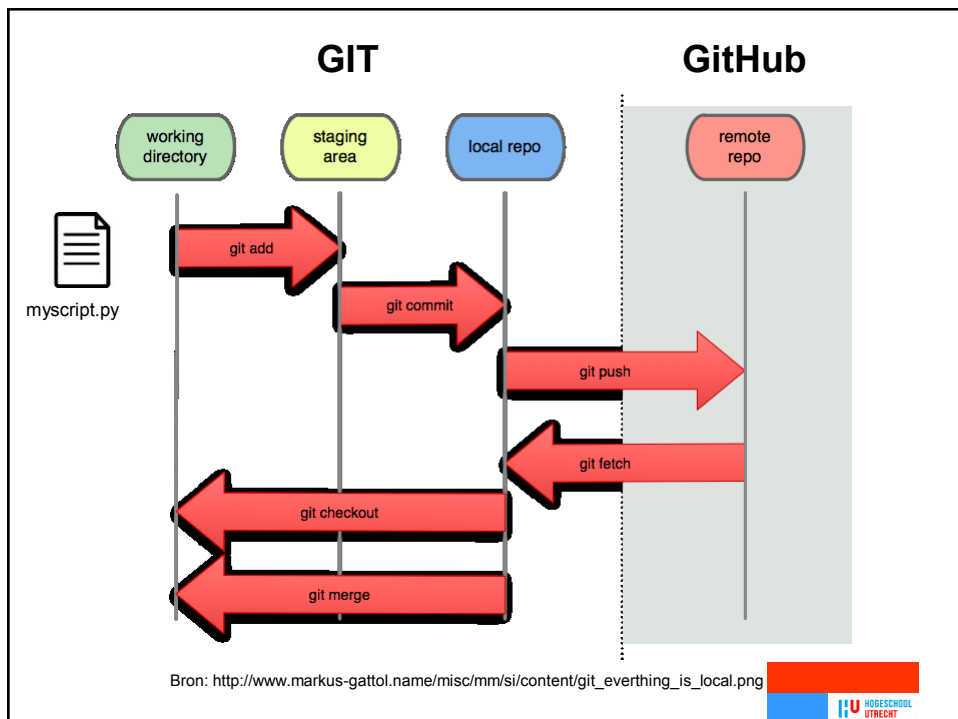
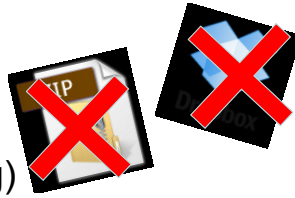
1 import datetime
2
3 class Person():
4     '''This class is about Persons'''
5
6     unique_number=0 # Every person gets a new unique number
7
8     def init(self, fullname, city, birth_year=1900):
9         '''Person constructor'''
10        self.fullname      = fullname
11        self.city           = city
12        self.birth_year     = birth_year
13        self.accountnumber  = None
14        self.number         = Person.unique_number
15        Person.unique_number += 1
16
17    def calculate_age():
18        '''Calculate age of person'''
19        return datetime.date.today().year - self.birth_year
20
21    def __str__(self):
22        '''Return a string with relevant information about this person'''
23        print("Naam: "+self.fullname)
24
25
26 p1=Person("Kees Jansma","Amersfoort",1966)
27 print(p1.calculate_age)
28 print(p1)

```

5 fouten

## Source Control: Why?

- Een **Back-up**
- **Label** je software (tagging)
- Zorg voor **historie**
- **Samen** werken aan dezelfde code



## Source Control: Demo

- git init
- git remote add origin `https://github.com/user/repo.git`
- git add <bestand.py>
- git commit -m "message"
- git push origin master



## Meer informatie...

### ▪ Namespaces:

- Perkovic Chapter 7 (7.1/7.2/7.4)
- Swaroop (Byte of Python): Chapter 9 (9.1/9.2/9.3)

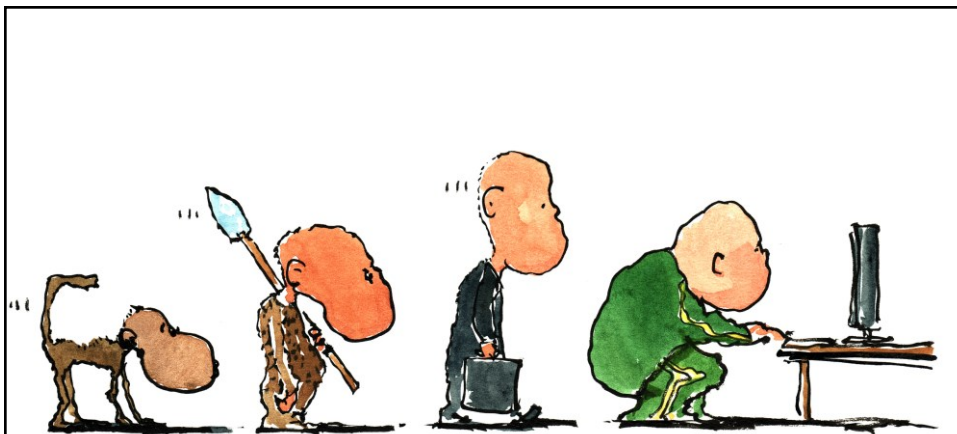
### ▪ Object Oriented Programming:

- Perkovic: Chapter 8
- Swaroop (Byte of Python): p. 85 t/m 95

### ▪ Version Control:

- <http://rogerdudler.github.io/git-guide/index.nl.html>
- <https://www.codeschool.com/courses/try-git>






[Man Chain illustration](#) by [Frits Ahlefeldt-Laurvig](#) | CC BY-ND

# Algoritmes

Programming (TICT-V1PROG-15)  
HBO-ICT propedeuse periode 1 2015-2016



## Algoritmes

- Sorteren
  - Bubble Sort
  - Insertion Sort

## Sorteren



[Sorting Skittles by Colour](#) by [Ben Watts](#) | [CC BY](#)

28



## Wat weten we al?

- Lijst sorteren:

```
stations = ['Utrecht Centraal', 'Geldermalsen',
            'Amsterdam Centraal']
stations.sort()
print(stations)
geeft
['Amsterdam Centraal', 'Geldermalsen', 'Utrecht
Centraal']
```

- Alternatief:

```
stations = ['Utrecht Centraal', 'Geldermalsen',
            'Amsterdam Centraal']
print(sorted(stations))
```

29



## Wat weten we al?

- Ander datatype sorteren:

```
traject = {'Utrecht Centraal': '18:41',  
'Geldermalsen': '19:06', '\s-Hertogenbosch':  
'19:25'}  
print(sorted(traject))  
geeft  
['\s-Hertogenbosch', 'Geldermalsen', 'Utrecht  
Centraal']
```

De sorteertechniek die hier achter zit, heet TimSort, vernoemd naar de bedenker ervan, Tim Peters. Voor meer info, zie <http://en.wikipedia.org/wiki/Timsort>

30



## Bubble sort

- Bekijk de uitleg op <https://youtu.be/8Kp-8OGwphY>

31



## Bubble sort in Python

```
def bubbleSort(lijst):
    for aantalNummers in range(len(lijst)-1, 0, -1):
        for i in range(aantalNummers):
            if lijst[i] > lijst[i+1]:
                lijst[i], lijst[i+1] = lijst[i+1], lijst[i]
            # wissel om

stations = ['Geldermalsen', 'Utrecht Centraal', 'Amsterdam
Centraal', '\s-Hertogenbosch']
bubbleSort(stations)
print(stations)
```

Tel terug van  $N - 1$ , naar  $N - 2$ , etc. tot je 0  
(lijst gesorteerd) bereikt

ga van 0 naar eind v/h  
als item groter dan volgende  
ongesorteerde deel

wissel om

32



## Bubble sort in Python (per stap)

```
['Geldermalsen', 'Utrecht Centraal', 'Amsterdam Centraal',
'\s-Hertogenbosch']

['Geldermalsen', 'Amsterdam Centraal', 'Utrecht Centraal',
'\s-Hertogenbosch']

['Geldermalsen', 'Amsterdam Centraal', '\s-Hertogenbosch',
'Utrecht Centraal']

['Amsterdam Centraal', 'Geldermalsen', '\s-Hertogenbosch',
'Utrecht Centraal']

['Amsterdam Centraal', '\s-Hertogenbosch', 'Geldermalsen',
'Utrecht Centraal']

['\s-Hertogenbosch', 'Amsterdam Centraal', 'Geldermalsen',
'Utrecht Centraal']
```

33





## Meer over Bubble sort

- <http://interactivepython.org/courselib/static/python/ds/SortSearch/TheBubbleSort.html>
- [http://nl.wikipedia.org/wiki/Bubblesort#Implementatie\\_in\\_Python](http://nl.wikipedia.org/wiki/Bubblesort#Implementatie_in_Python)

34



## Insertion sort

- Bekijk de uitleg op <https://youtu.be/DFG-XuyPYUQ>

35



## Stappenplan

- ga van links naar rechts
  - als item kleiner is dan item links daarvan, wissel om
  - kijk of het (kleinere) item nog verder naar link geschoven kan worden (= herhaal vorige stap)
- Zo bouw je een steeds groter wordende (linker) deellijst op die gesorteerd is
- De waarde aan de rechterkant wordt als het ware op de juiste plek gezet aan de linkerkant

Les 9

36



## Insertion sort (voorbeeld)

```
['Geldermalsen', 'Amsterdam Centraal', 'Utrecht Centraal', "'s-Hertogenbosch"]
['Amsterdam Centraal', 'Geldermalsen', 'Utrecht Centraal', "'s-Hertogenbosch"]
['Amsterdam Centraal', 'Geldermalsen', "'s-Hertogenbosch", 'Utrecht Centraal']
['Amsterdam Centraal', "'s-Hertogenbosch", 'Geldermalsen', 'Utrecht Centraal']
["'s-Hertogenbosch", 'Amsterdam Centraal', 'Geldermalsen', 'Utrecht Centraal']
["'s-Hertogenbosch", 'Amsterdam Centraal', 'Geldermalsen', 'Utrecht Centraal']
```

37



## Opdracht (15 min): Insertion sort toepassen

```
stations = ['Geldermalsen', 'Utrecht Centraal',
            'Amsterdam Centraal', '\s-Hertogenbosch']
```

- Sorteer volgens insertion sort 'pseudocode':

```
for i ← 1 to length(A) - 1  gebruik hier range()
    j ← i
    while j > 0 and A[j-1] > A[j]
        swap A[j] and A[j-1]  neem bubble sort als voorbeeld
                               voor het omwisselen
        j ← j - 1
    end while  beide 'end' zijn niet nodig.
end for
```

Les 9

38



## Meer over Insertion sort

- [http://en.wikipedia.org/wiki/Insertion\\_sort](http://en.wikipedia.org/wiki/Insertion_sort)
- <http://interactivepython.org/courselib/static/pythonds/SortSearch/TheInsertionSort.html>
- [http://runnable.com/Uq\\_i4quWT8lOAAAv/algorithms-insertion-sort-python](http://runnable.com/Uq_i4quWT8lOAAAv/algorithms-insertion-sort-python)

Les 9

39



## Weekopdracht:

Maak thuis de opdrachten die horen bij:

- Week 6: College 1
- Week 6: Weekopdracht

40



## Volgende les:

- Tentamen!



<http://spitswww.uvt.nl/studyskills/images/multiple-choice-tentamen.jpg>

41

