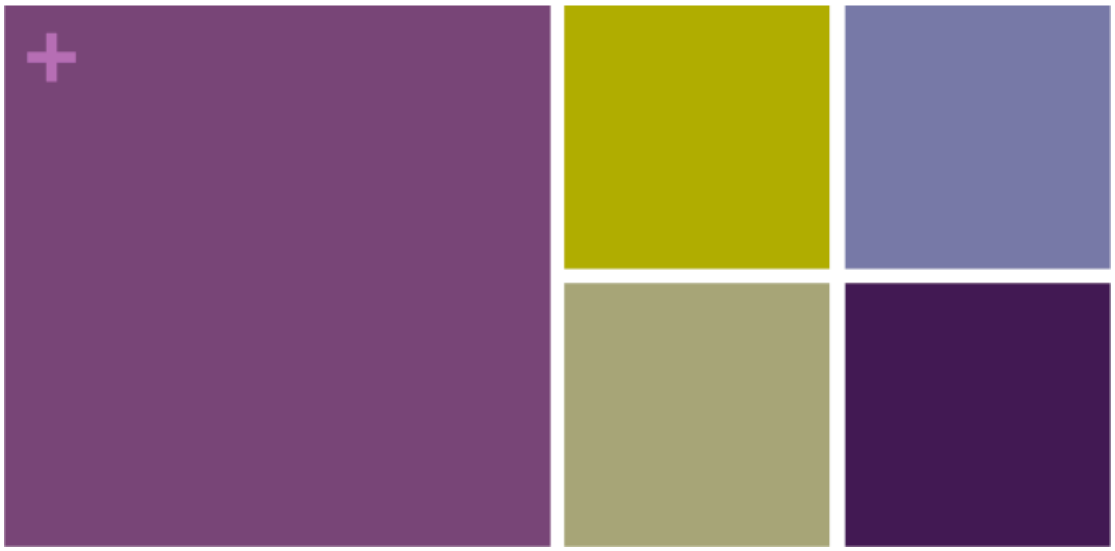
Informatica



Deel I: Python (12 HOC)

Prof. Dr. Ann Dooms

Inhoudstafel

Les 1

- Statements
  - Uitdrukking/Expression
  - Toekenning/Assignment
  - Functies (definitie en call)

- Types, operatoren en built-in functies
  - int, float
  - str
  - list
  - tuple
  - bool

Les 2

- Modules
- Objecten en methodes

Les 3

- Control flow statements
  - Keuzes maken: if
  - Iteraties
    - Counted loops: for
    - Conditional loops: while
  - Iteraties onder controle houden
  - Fouten opsporen, onder controle houden en debuggen

Les 4

- Recursie
- Werken met bestanden/files

Les 5

- Algoritmes
  - Zoeken/Search
    - Linear search
    - Binary search
  - Sort/Sorteren
    - Selection en insertion sort
    - Merge sort

Les 6

- Python Wrap Up
- Een laatste algoritme van naaldje tot draadje

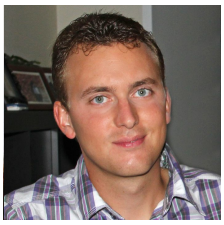
Who is Who?

Ann & Jan

Prof. Dr. Ann Dooms – ann.dooms@vub.ac.be - PL9.2.29
Prof. Dr. Jan Lemeire – jan.lemeire@vub.ac.be - PL9.2.28 & KE.3.08
Vakgroep Elektronica en Informatica (ETRO)

Assistenten

Wesley, Steffen, Bruno & Bob



Wesley Mattheyses

wmatthey@etro.vub.ac.be
Ke.3.13

Steffen Thielemans

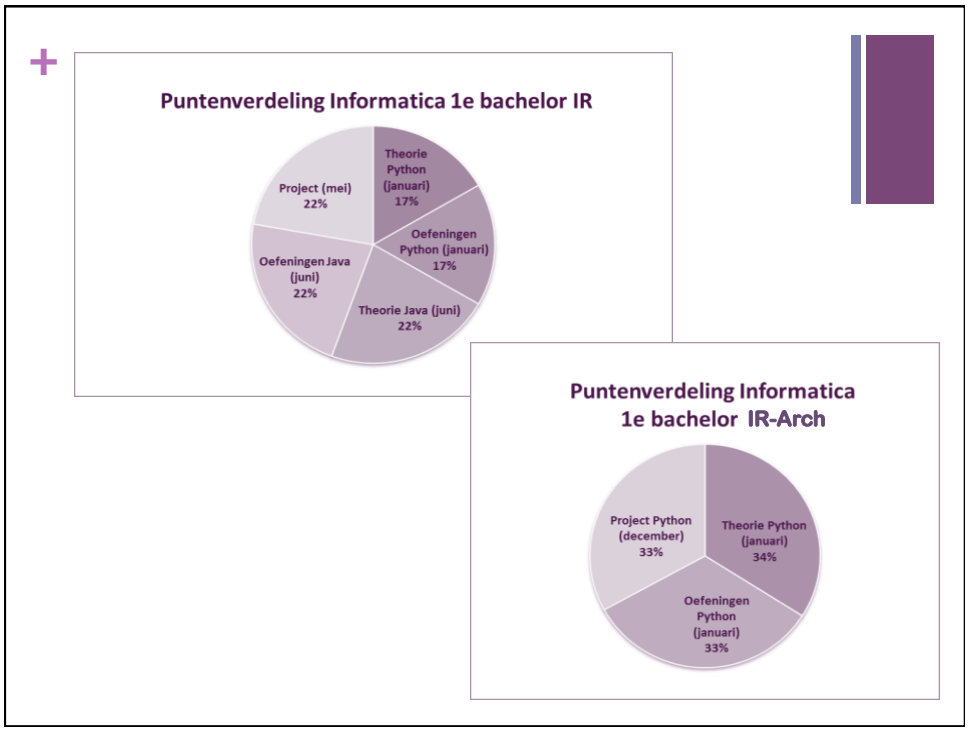
sthielem@etro.vub.ac.be
K.4.209

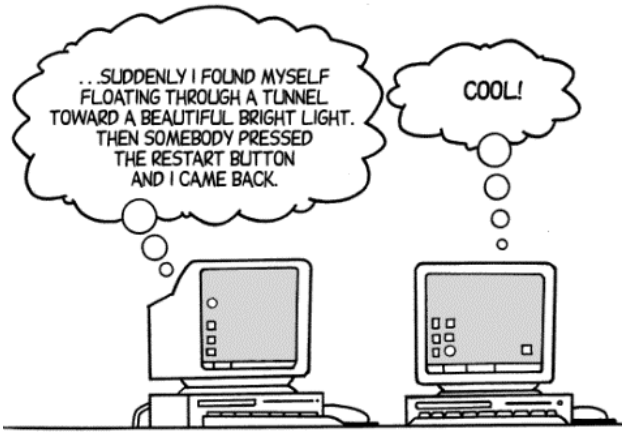
Bob Andries

bandries@etro.vub.ac.be
PL.9.26

Bruno Cornelis

bcorneli@etro.vub.ac.be
PL.9.27

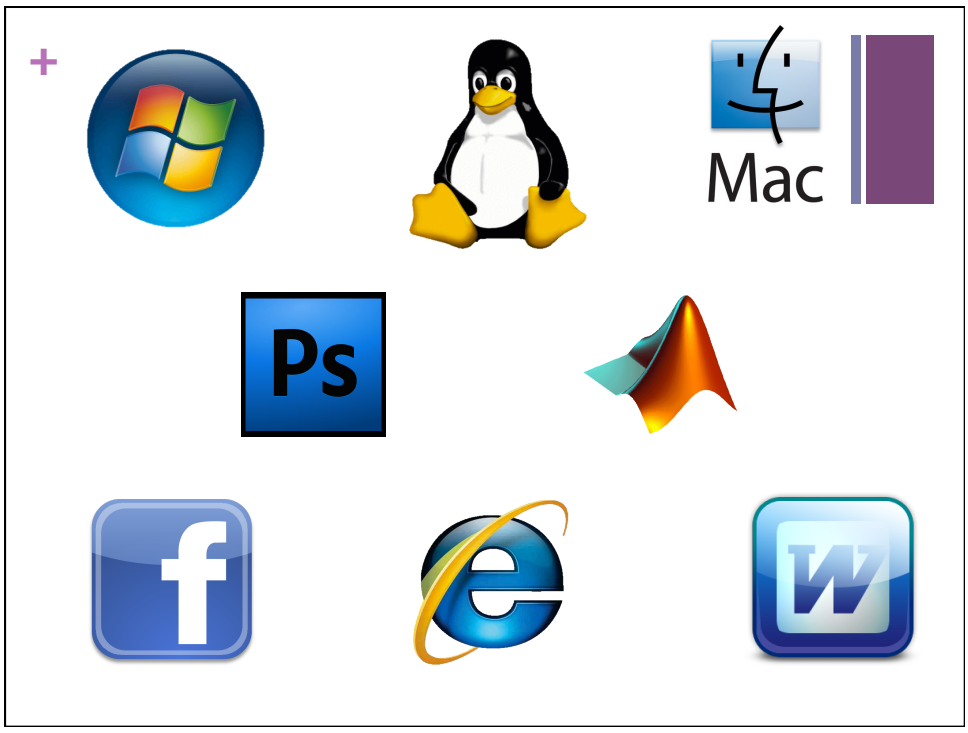

Puntenverdeling Informatica 1e bachelor IR
Theorie Python (januari) 17%
Oefeningen Python (januari) 17%
Theorie Java (juni) 22%
Oefeningen Java (juni) 22%
Project (mei) 22%
Puntenverdeling Informatica 1e bachelor IR-Arch
Theorie Python (januari) 34%
Oefeningen Python (januari) 33%
Project Python (december) 33%


...SUDDENLY I FOUND MYSELF FLOATING THROUGH A TUNNEL TOWARD A BEAUTIFUL BRIGHT LIGHT. THEN SOMEBODY PRESSED THE RESTART BUTTON AND I CAME BACK.
COOL!

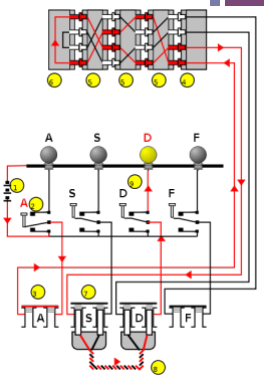
Informatica

Les 1

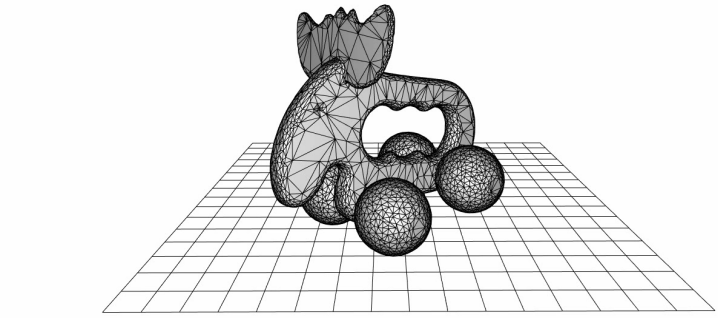
Computers & Co


Mac
Ps

De Enigma code kraken



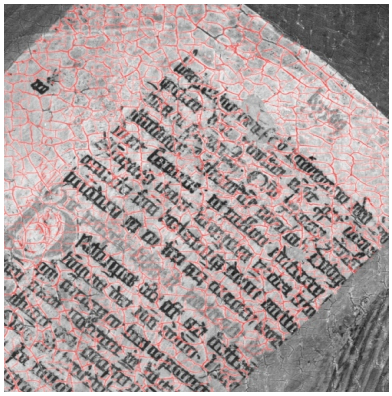
Meshes: representatie van 3D-objecten



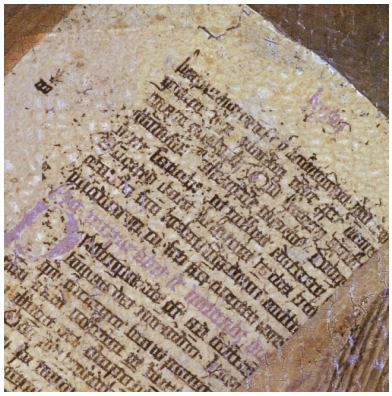
http://closertovaneyck.kikirpa.be

Digitale Schilderij-Analyse

Craquelure detectie



Restoratie door inpainting



Programma, Shell & Compiler

2.1

- Een *programma* is een verzameling instructies.
- Een *shell* is een interactief programma waarmee een gebruiker met een *Command Line Interface* (>>> wordt de *prompt* genoemd) opdrachten kan geven aan het besturingssysteem (e.g. Windows, Mac OS, …) van een computer.
- Een *compiler* (letterlijk samensteller of opbouwer) is een computerprogramma dat een invoer vertaalt in een bepaalde uitvoer.

Programmeertalen

Modula-2 C

C++ PHP

Python Fortran

Lisp PL1

Scheme

C# Pascal

VisualBasic Java

Ada

JavaScript

en nog vele andere ...

Ruby

Python

Python

- Taal:

  http://www.python.org (versie 2.x)

- Editor:

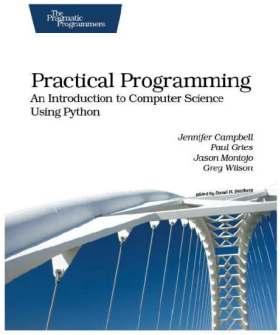
  Wing 101 IDE (http://www.wingware.com)

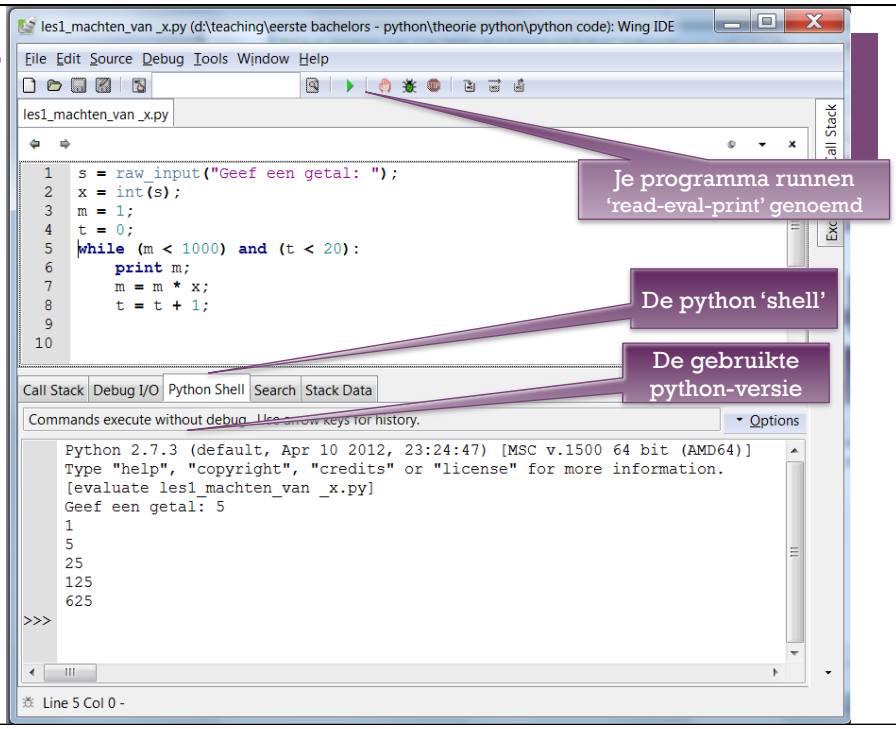
- Cursus website:

  http://www.etro.vub.ac.be/Education/Courses/Python/

- Handboek (VUBtiek):

  Practical Programming
  An introduction to Computer Science using Python
  Jennifer Campbell, Paul Gries, Jason Montojo, Greg Wilson
  Hoofdstukken 1-8, 10, 11





Statements

Instructies / Statements

- Uitdrukking (*Expression*)

```
>>> 4 + 13
17
```

- Toekenning (*Assignment*)

```
>>> degrees_celsius = 26.0
```

- Functie

```
>>> def to_celsius(t):
. . .       return (t - 32.0) * 5.0 / 9.0
. . .
```

Expression

- Bestaat uit *waarden* en *operatoren* (+, -, /, *, %, **, …):

```
>>> 4 + 13
```

Gehele waarde (int)

Optelling

- Waarden hebben steeds een *type* en bijhorende *operatoren*:
  - getallen (gehelen/integers, floating points)
  - string
  - lijst, tupel
  - Boolean
  - NoneType
- Een expression wordt uitgevoerd in bepaalde *orde*.

Assignment/Variabele

- Een *variabele* is een naam voor een waarde in het geheugen:

```
>>> degrees_celsius = 26.0
```

Variabele

Assignment

- De waarde van de variabele is variabel.
- Een variabele heeft een *bereik* (*scope*).
- Wanneer twee variabelen refereren naar eenzelfde waarde, noemen we ze *aliases*.

Functie - Definitie

- Een functie neemt een waarde als input en geeft een waarde als output.
- Een functie wordt gedefinieerd als volgt:

```
def function_name(parameters):
    block
```

- Een *parameter* is een (lokale) variabele.
  Een functie kan geen of meerdere parameters hebben.
- Een functie groepeert veel voorkomende of ingewikkelde statements in *block*, waarbij return expression de output waarde van de functie is.

```
>>> def to_celsius(t):
. . .     return (t - 32.0) * 5.0 / 9.0
. . .
```

Parameter

- Python heeft ook enkele built-in functies.

Functie - Call

- Een functie wordt aangeroepen door de parameters een waarde te geven die we *argument* noemen.
- De statements binnen de functie definitie worden uitgevoerd met de gekozen waarden

```
>>> to_celsius(212)
100.0
```

Opmerking

Een operator is een mooie syntactische voorstelling van een functie waarbij de operandi de argumenten zijn.

Types

Type int

Het type *int* kan alle mogelijke integer waarden aannemen tussen -2^{32} en 2^{32}.

Type float

Het type *float* stelt fractionele getallen (*floating point*) voor door een punt te gebruiken.

Type str

- Een *string* is een opeenvolging van karakters, i.e. letters, getallen en symbolen.
  - str
  - unicode
- Kan worden gecreëerd door het plaatsen van ‘ ’ of “ ”.
- Lege (*empty*) string: ‘ ’
- Er zijn speciale karakters voor bvb. een nieuwe lijn (/n) en tab (/t).

Type list

- Een *lijst* is leeg of houdt meerdere objecten bij van om het even welk soort type in een welbepaalde orde.

```
>>> [‘a’,‘b’,‘c’]
```

- Lege (*empty*) lijst: []
- De objecten worden *elementen* genoemd.
- Je kan naar de elementen refereren aan de hand van hun positie of *index* die loopt van 0 tot (lengte lijst)-1.
- Een lijst is *muteerbaar*, i.e. de inhoud kan veranderd worden.

Type tuple

- Een *tupel* is leeg of houdt meerdere objecten bij van om het even welk soort type in een welbepaalde orde.

```
>>> ('a','b','c')
```

- Een tupel is *niet muteerbaar*, i.e. de referenties naar de objecten kunnen niet veranderd worden.

Opmerking

Zowel op str als tupel kunnen de list operatoren en built-in functies worden toegepast.

Type bool

Dit type heeft maar *twee* mogelijke waarden:

- False
- True

Assignment

- Voorbeelden
  - x = True
  - a = [5, 6]
  - b = []
- Dit mag niet!
  - x = x + y (x en y zijn niet gekend)
- Variabele krijgt type bij toewijzing
  - degrees_celsius = 26 (int)
  - degrees_celsius = 26.0 (float)
    - Verschil = 100 - degrees_celsius wordt ook een float

Hergebruik van zelfde variabelenaam

- Voorbeeld
  - x = 5
  - x = [x, 5]
- Je mag veranderen van type!!
- Niet aan te raden, onduidelijk!

Operatoren

Numerieke operatoren (int & float)

p. 23

1) -: negatie
2) **: machtsverheffing
3) *: vermenigvuldiging, /: deling, %: rest bij deling
4) +: optelling, -: aftrekking

Precedence (volgorde van bewerking)

Voorbeeld:

```
>>> 4 + 3 * -2**2
```

=> resultaat ??

Combined operators

- +=: optellen bij variabele
  - x += 2; is identiek aan x = x + 2;
- -=, /=, *=: analoog

Berekeningen met integers

OPGEPAST

- Alles wat achter de komma is gaat weg!!

```
>>> 17/10
1
```

- Gebeurt als je werkt met integers

```
>>> x = 5
>>> y = x / 2
>>> print y
2
```

- Ook in de tussenresultaten!!

```
>>> 5/2*2
4
```

- Voorkomen: aangeven dat je werkt met floats

```
>>> 17.0/10
1.7
>>> x = 5
>>> y = x / 2.0
>>> print y
2.5                    (y is nu een float)
```

Numerieke built-in functies

- abs(x): absolute waarde van een getal
- float(x): zet een getal om in een getal van type float
- int(x): zet een getal om in een getal van type int door staart af te kappen
- round(x): zet een getal om naar dichtsbijzijnde integer zonder type te wijzigen
- pow(x, y): de macht x^y nemen (idem als **-operator)

String operatoren

Creëren een nieuwe string:

- +: concatenatie
- * geheel getal: herhaling

Opmerking

\+ is een *overloaded* operator, i.e. een verschillend gedrag naargelang type waarop hij wordt toegepast.

String built-in functies

- str(x): converteert x naar een string
- len(x): lengte van een string
- De built-in functie raw_input geeft de gebruiker's input de waarde van een string.

Lijst operatoren

Creëren een nieuwe lijst:

- +: concatenatie

```
>>> [1, 2] + [3, 4]
[1, 2, 3, 4]

>>> [1, 2] + 3
TypeError: can only concatenate list (not "int") to list
```

- * geheel getal: herhaling

```
>>> [1, 2] * 3
[1, 2, 1, 2, 1, 2]
```

- *Slicing* (list[i:j]) creëert een nieuwe lijst:

```
>>> L[0:4]
```

Lijst built-in functies

Zij L een lijst, dan

- len(L): lengte lijst, i.e. het aantal elementen
- max(L), min(L): grootste, resp. kleinste waarde in L
- sum(L): som van de (numerieke) waarden in L (cfr. p81)

De range-functie

range(start, stop, step size)

- Genereert een lijst van gehele getallen vertrekkende vanaf start tot stop -1, met step size tussen elke waarde.

```
>>> range(2, 10, 2)
[2, 4, 6, 8]
>>> range(2, 10, -2)
[]
>>> range(20, 10, -2)
[20, 18, 16, 14, 12]
```

- Je kan range ook aanroepen met twee argumenten: step size is dan 1.

```
>>> range(3, 7)
[3, 4, 5, 6]
```

- Je kan range ook aanroepen met één argument: start is dan 0.

```
>>> range(4)
[0, 1, 2, 3]
```

Boolean operatoren

- or

or	False	True
False	False	True
True	True	True

- and

and	False	True
False	False	False
True	False	True

- not

not	False	True
	True	False

Relationele operatoren

- Operatoren
  - >: groter dan
  - <: kleiner dan
  - >=: groter dan of gelijk aan
  - <=: kleiner dan of gekijk aan
  - ==: gelijk aan
    - '=' staat voor toekenning!
  - !=: niet gelijk aan
- **Resultaat: boolean!!**
  - Kan je dus toewijzen aan een booleaanse variabele (6.3)

```
>>> young = age < 45
>>> slim = bmi < 22.0
```

Volgorde van bewerkingen

- *Numerieke* operatoren hebben hogere orde dan *relationele*, welke hoger zijn dan *Booleaanse* operatoren.

```
>>>  2 * 2 < 3 or 4 - 2 > 3
```

=> *resultaat ??*

- Beter, duidelijker: haakjes zetten, ook als niet nodig!

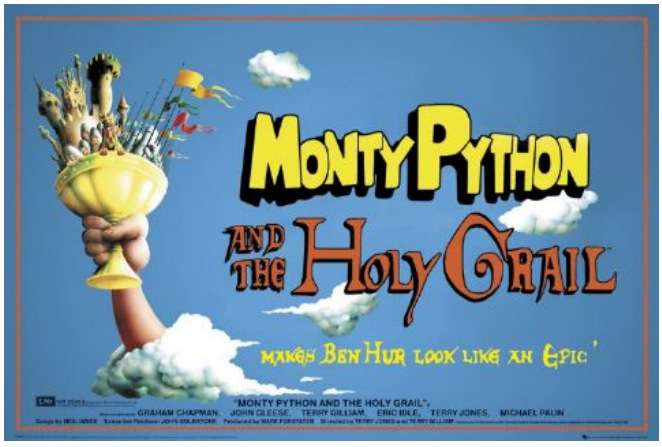
```
>>>  ((2 * 2) < 3) or ((4 – 2) > 3)
```

Oops …

Informatica

Les 2

Python



Statements

Instructies / Statements

- Uitdrukking (*Expression*)

```
>>> 4 + 13
17
```

- Toekenning (*Assignment*)

```
>>> degrees_celsius = 26.0
```

- Functie

```
>>> def to_celsius(t):
. . .       return (t - 32.0) * 5.0 / 9.0
. . .
```

Expression

- Bestaat uit *waarden* en *operatoren* (+, -, /, *, %, **, …):

```
>>> 4 + 13
```

Gehele waarde (int)

Optelling

- Waarden hebben steeds een *type* en bijhorende *operatoren*:
  - getallen (gehelen/integers, floating points)
  - string
  - lijst, tupel
  - Boolean
  - NoneType
- Een expression wordt uitgevoerd in bepaalde *orde*.

Assignment/Variabele

- Een *variabele* is een naam voor een waarde in het geheugen:

```
>>> degrees_celsius = 26.0
```

Variabele

Assignment

- De waarde van de variabele is variabel.
- Een variabele heeft een *bereik* (*scope*).
- Wanneer twee variabelen refereren naar eenzelfde waarde, noemen we ze *aliases*.

Functie - Definitie

- Functies helpen bij berekeningen of het uitvoeren van opdrachten.
- Een functie neemt een waarde als input en geeft een waarde als output.
- Een functie wordt gedefinieerd als volgt:

```
def function_name(parameters):
    block
```

- Een *parameter* is een (lokale) variabele.
  Een functie kan geen of meerdere parameters hebben.

Voorbeeld

- Een functie groepeert veel voorkomende of ingewikkelde statements in *block*, waarbij return expression de output waarde van de functie is.

```
>>> def to_celsius(t):
...             return (t - 32.0) * 5.0 / 9.0
...
```

Parameter

- Python heeft ook vele built-in functies.

Functie - Call

- Een functie wordt aangeroepen door de parameters een waarde te geven die we *argument* noemen.
- De statements binnen de functie definitie worden uitgevoerd met de gekozen waarden

```
>>> to_celsius(212)
100.0
```

Opmerking

Een operator is een mooie syntactische voorstelling van een functie waarbij de operandi de argumenten zijn.

Modules

Modules

- Een *module* is een collectie van functies en variabelen (die gerelateerd zijn) in een .py file.
- Een module wordt geimporteerd via *import.*

```
>>> import math
```

- De inhoud kan worden aangeroepen door

```
>>> module_naam.ding_naam(parameterlist)
```

- Modules kunnen geimporteerd worden binnen andere modules of programma's. Een module met modules noemt men een *package*.

Modules

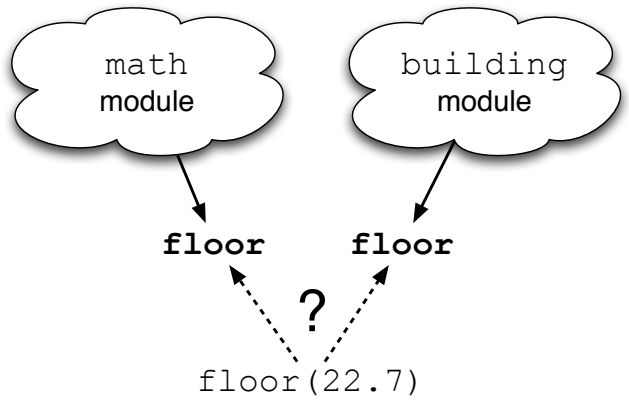

```
>>> import math
>>> sqrt(9)
Traceback (most recent call last):
  File "<string>", line 1, in <fragment>
NameError: name 'sqrt' is not defined
>>> math.sqrt(9)
3.0
>>> from math import sqrt
>>> sqrt(9)
>>> from math import *
```

eerste manier van oproepen

tweede manier

alle functies importeren

Modules



Modules


```
>>> help(math)
Help on module math:

NAME
    math

FILE
    /Library/Frameworks/Python.framework/Versions/2.7/lib/
python2.7/lib-dynload/math.so

MODULE DOCS
    http://docs.python.org/library/math

DESCRIPTION
    This module is always available.  It provides access to the
    mathematical functions defined by the C standard.
```

Modules

FUNCTIONS

acos(...)

acos(x)

Return the arc cosine (measured in radians) of x.

…

DATA

e = 2.718281828459045

pi = 3.141592653589793

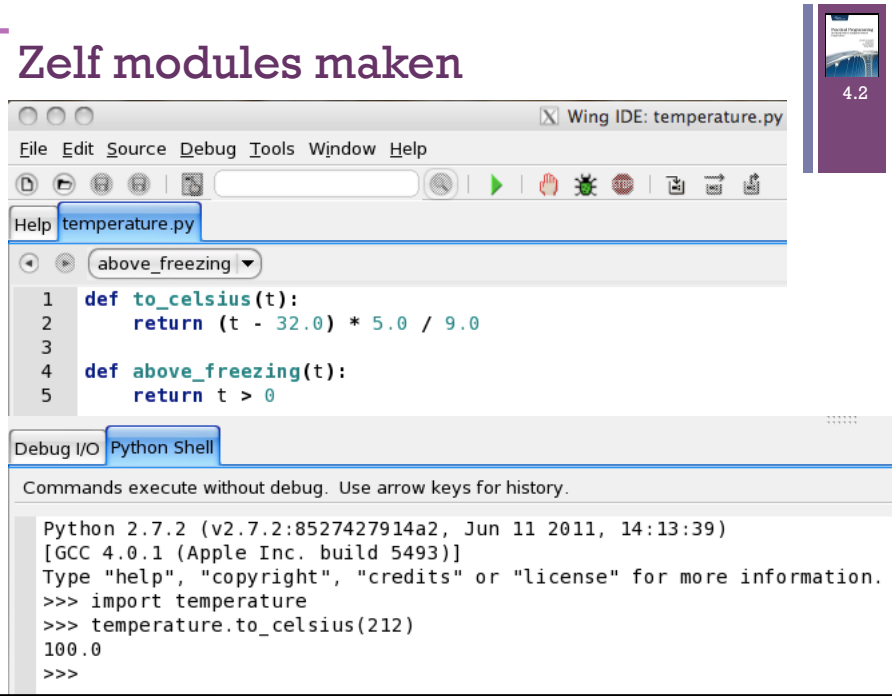
Standard Python library

- Dit is een bibliotheek die verschillende honderden modules bevat.
- De volledige lijst is te vinden op

  http://docs.python.org/modindex.html.
- Je kan zulke modules of packages installeren op je computer.

  Voorbeeld: Python Imaging Library (PIL)
- Python's built-in functies zijn terug te vinden in de module __builtins__

Zelf modules maken



Zelf modules maken

Documenteer steeds je module, evenals de functies en variabelen die ze bevat via *documentation strings* (docstrings).

```
'Functions for working with temperatures.'

def to_celsius(t):
    'Convert Fahrenheit to Celsius.'
    return round((t - 32.0) * 5.0 / 9.0)

def above_freezing(t):
    'True if temperature in Celsius is above freezing, False otherwise.'
    return t > 0
```

Objecten en Methodes

Objecten en Methodes

Object-geörienteerd

- Je zou kunnen alle functies die we op strings willen toepassen groeperen in een module, denk aan
  - maken van hoofdletters
  - blanco's deleten
  - een stuk vervangen
  - …
- Er is ook een andere oplossing: je kan functies maken behorende bij een specifiek type die meteen toepasbaar zijn op al de waarden.
- Dit noemt met *object-georiënteerd* programmeren.

Objecten en Methodes

- Een waarde in Python (getallen, strings, tupels, maar ook functies, matrices, …) noemt men een *object*. Het heeft een bepaald type en een plaats in het geheugen.
- Objecten bevatten *methodes*, i.e. speciale type-afhankelijke functies die in objecten zitten en die je aanroept met een "." (je kan hiermee ook ketens maken).

```
object.method_naam(parameterlist)
```

- Operatoren zijn eigenlijk mooie syntactische voorstellingen voor methodes. De read-fase doet het "vuile werk".

Opmerking

Na het importeren van een module, nemen we de functies en variabelen hierbinnen ook vast met een "."

String Methodes

Elke string heeft automatisch alle methodes die behoren tot het str type.

Voorbeelden:

- string.capitalize()
- string.find(waarde)
- string.split()

Opmerking

- Je vindt de volledige lijst online of door help(str) te typen.
- Getallen en strings zijn immutable. Hun methodes creëren nieuwe getallen en strings, indien nodig.

Lijst Methodes

Ook elke lijst is een object en heeft dus de methodes die behoren tot het list type.

- list.append(waarde)
- list.insert(index, waarde)
- list.remove(waarde)
- list.reverse()
- list.sort()

Opmerking

Lijsten zijn mutable. Hun methodes veranderen meestal de lijst in plaats van een nieuwe te creëren.

Voorbeelden

- Python kan ook werken met beelden, geluid en video. Deze objecten hebben hun eigen methodes.
- Python Imaging Library (PIL)

```
>>> import Image
>>> Image.open('MonumentValley.jpg').show()
```

Opmerking

show() roept extern programma aan om de afbeelding te tonen.

Chain method calls

p. 62-63

```
>>> 'Computer Science'.swapcase()
'cOMPUTER sCIENCE'
>>> 'Computer Science'.swapcase().endswith('ENCE' )
True
```

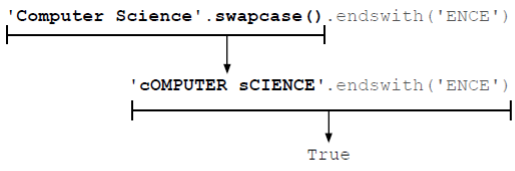


Figure 4.4: Chaining method calls

Informatica

Les 3

Statements

Instructies / Statements

Expressions, *assignments* en *functies* zijn lang niet alle *statements* die Python begrijpt!

We zagen ook al:

- return : laat een functie de waarde van de bijhorende expressie teruggeven (zie Les 1, hoofdstuk Instructies, definitie functie)
- import module/ from module import : laadt een module (zie Les 2, hoofdstuk Modules)

Opmerking

We kunnen ook expliciete output voor de gebruiker genereren met het zogenaamde print statement.

Control Flow Statements

Kunnen we met de statements die we tot nu toe ingevoerd hebben alle wiskundige functies definiëren?

Neem de functie *absolute waarde:*

$$abs(x) = \begin{cases} x & x \geq 0 \\ -x & x < 0 \end{cases}$$

Neen! Dit is een *stuksgewijze/vertakte* functie waarbij we een *keuze* maken.

Control Flow Statements

Wat als we een taak een aantal keer willen uitvoeren?

"Ik wil de naam veranderen van al mijn foto's in mijn folder van DSCx.jpg naar USAx.jpg."

Wat als we een taak willen uitvoeren zolang een voorwaarde is voldaan?

"Blijven rondjes lopen totdat Evy Gruyaert STOP zegt!"

Keuzes maken: if statement

if statement

Een *if statement* heeft de vorm

```
if condition:
        if-block
else:
        else-block
```

clause

- condition : een expressie die True of False is
- if-block/else-block : een lijst van Python statements (zoals de *block* bij functies, enkel *return* is niet toegelaten)

if statement

- Wanneer de *condition True* is, wordt de *if-block* uitgevoerd.
  Wanneer de *condition False* is, wordt de *else-block* uitgevoerd.
- Je hoeft *else* niet te gebruiken.
  Wanneer de *condition False* is, gebeurt er dan niets.
- Soms is een enkele beslissing niet genoeg en moeten meerdere criteria onderzocht worden.
  - meerdere if
  - meerdere "else if", i.e. elif

```
elif condition:
        elif-block
```

- Er kan wel maar één *else clause* zijn en die komt op het einde.
  Wordt enkel uitgevoerd als alle voorgaande testen False geven.

Control flow if/elif statements

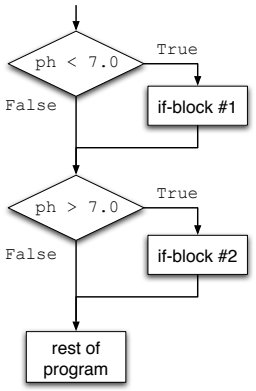


Figure 6.4: if statement

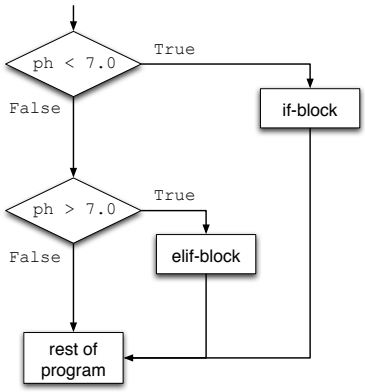


Figure 6.5: elif statement

Geneste if statements

Een *if-block* kan natuurlijk ook *if* statements bevatten.

```
input = raw_input()
if len(input) > 0:
    ph = float(input)
    if ph < 7.0:
        print "%s is acidic." % (ph)
    elif ph > 7.0:
        print "%s is basic." % (ph)
    else:
        print "%s is neutral." % (ph)
else:
    print "No pH value was given!"
```

outer

inner

We spreken van inner en outer *if* statements.

Iteraties: for loop statement

Counted loop

for statement

5.4
7.1

Een *for loop statement* heeft de vorm

```
for variable in seq:
    block
```

- variable : een variabele
- seq : een lijst, tupel of string
- block : een lijst van Python statements
  (zoals de *block* bij functies, enkel *return* is niet toegelaten)

for loop statement

- Python voert de *block* uit voor iedere waarde in de lijst.
- Elk zulke stap noemen we een iteratie.
- Bij het begin van elke iteratie voert Python een *assignment* door van de variabele aan het volgende element uit de lijst.
- Als de variabele reeds een waarde had voor de for loop, wordt deze gewoon overschreven. De variabele behoudt echter wel de laatste waarde uit de iteratie.

for loop statement


```
>>> velocities = [0.0, 9.81, 19.62, 29.43]
>>> for v in velocities:
...     print "Metric:", v, "m/sec;",
...     print "Imperial:", v * 3.28, "ft/sec"
...

>>> for c in 'alpha':
...     print c
...
```

Geneste for loop statements

Een for loop-block kan natuurlijk ook *for loop* statements bevatten.

```
>>> outer = ['Li', 'Na', 'K']
>>> inner = ['F', 'Cl', 'Br']
>>> for metal in outer:
...     for gas in inner:
...         print metal + gas
...
...
LiF
LiCl
LiBr
NaF
NaCl
NaBr
KF
KCl
KBr
```

outer

inner

We spreken van inner en outer *for loop* statements.

Hulpmiddelen bij for loops

Wat als we de elementen uit de lijst zelf willen vervangen?

Voorbeeld:
Stel dat we de elementen uit een lijst L willen verdubbelen.

- L[0]=2*L[0]
  L[1]=2*L[1]
  …

Probleem met de lengte van L!

- for i in L:
  i=2*i

Dit wijzigt L niet!

Hulpmiddelen bij for loops

- Gebruik de built-in functie range:

  range(start, stop, step size)

- Voorbeeld:

```
for i in range(len(L)):
    L[i]=2*L[i]
```

Hulpmiddelen bij for loops

- Gebruik de built-in functie enumerate:

  enumerate(x)

  waarbij x is een lijst, tupel of string is.

- Geeft tupel paren terug met als eerste element de index en als tweede element de waarde uit x op die index plaats.

- Gebruik multivalued assignments:

```
>>> x, y = 1, 2
>>> x
1
>>> y
2
```

```
>>> first, second, third = [1, 2, 3]
>>> first, second, third = 'abc'
```

Voorbeeld:

```
for (i,v) in enumerate(L):
    L[i]=2*v
```

Veranderen van elementen

Wat als we elementen uit een lijst zelf willen veranderen?

Voorbeeld: *verdubbelen elementen uit een lijst L.*

- Oplossing 1

```
L[0]=2*L[0]
L[1]=2*L[1]
…
L[9]=2*L[9]
```

Werkt enkel met met een vaste lengte van L (hier: 10)! En kan lange code worden.

- Oplossing 2

```
for i in L:
    i=2*i;
```

Dit wijzigt L niet, enkel i!

- Oplossing 3

```
for i in range(len(L)):
    L[i] = L[i]*2;
```

OK

Verwijderen van elementen

- Elementen verwijderen. Wat gebeurt er?

```
for i in l:
    l.remove(i)
```

- *Oplossing*: maak een copie van de lijst over dewelke je itereert en verwijder uit originele lijst.

```
for item in mylist[:]:
    mylist.remove(item)
```

Maakt kopie van de hele array

- **Conclusie**
  - For element in list: voornamelijk te gebruiken om elementen van een lijst te lezen

Iteraties: while loop statement

Conditional loop

while loop statement

- Een for loop is heel handig als je op voorhand weet hoeveel iteraties je nodig hebt
- Wanneer het aantal iteraties onbekend is, komt de while loop ter hulp.
- Een while loop zal blijven itereren zolang een voorwaarde is voldaan.

while loop statement

Een *while (of conditional) loop statement* heeft de vorm

```
while condition:
        block
```

- condition : een expressie die True of False is
- block : een lijst van Python statements
  (zoals de *block* bij functies, enkel *return* is niet toegelaten)

while loop statement

- Python voert de *block* zolang de voorwaarde voldaan is (condition=True) .
- Bij elke iteratie wordt de voorwaarde opnieuw gecontroleerd.
- Wanneer de voorwaarde niet meer voldaan is (condition=False), slaat Python de block over en stopt de loop.

while loop statement


```
>>> rabbits = 3
>>> while rabbits > 0:
...     print rabbits
...     rabbits -= 1
...
3
2
1
```

while loop statement

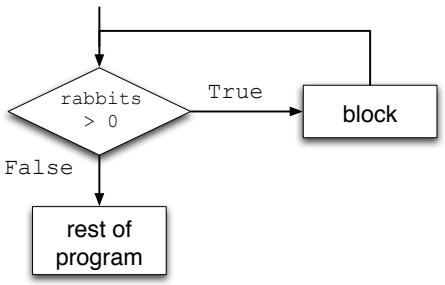


Figure 7.4: while loops

Testoefening

Wat is de output van het volgend programma?

```
x1 = 3
x2 = int(raw_input("geef een waarde "))  # hier tik ik 7 in
while x2 < 30:
    x3 = x1 + x2
    if (x3 % 3) == 0: # % is de rest operator (modulo), hier:
                                    rest bij deling door 3
        x2 = x2 + x3
    else:
        x2 = x2 - 1
print "De waarde van x1 is ", x1
print "De waarde van x2 is ", x2
print "De waarde van x3 is ", x3
```

Iteraties: loops onder controle houden

Forever and ever ∞


```
while True:
    formula = raw_input("Please enter a chemical formula: ")
    if formula == "H2O":
        print "Water"
    elif formula == "NH3":
        print "Ammonia"
    elif formula == "CH3":
        print "Methane"
    else:
        print "Unknown compound"

text = ""
while text != "quit":
    text = raw_input("Please enter a chemical formula (or 'quit' to exit): ")
    if text == "quit":
        print "...exiting program"
    elif text == "H2O":
        print "Water"
    elif text == "NH3":
        print "Ammonia"
    elif text == "CH3":
        print "Methane"
    else:
        print "Unknown compound"
```

Hoe stoppen?

Voorbeeld: bacteriën kweken

- Groei populatie met de tijd: P(t+1)=P(t)+rP(t)
- Hoe lang duurt het om de populatie te verdubbelen?

```
def wrong_grow(init_pop, growth_rate):
    time = 0
    pop = init_pop
    while pop != init_pop * 2:
        pop = pop + pop * growth_rate
        time = time + 1
        print pop
    return time
```

```
>>> wrong_grow(1000,0.21)
1210.0
1464.1
1771.561
2143.58881
2593.7424601
3138.42837672
3797.49833583
4594.97298636
5559.91731349
...
```

```
def grow(init_pop, growth_rate):
    time = 0
    pop = init_pop
    while pop < init_pop * 2:
        pop = pop + pop * growth_rate
        time = time + 1
        print pop
    return time
```

```
>>> grow(1000,0.21)
1210.0
1464.1
1771.561
2143.58881
4
```

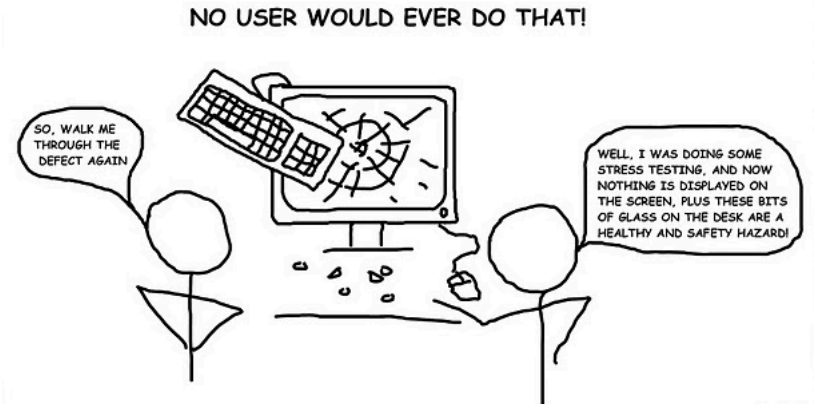
Infinite loop

Fouten opsporen, onder controle houden en debuggen

__name__

- Python definiëert een speciale variabele __name__ in elke module.
- Wanneer de module het hoofdprogramma is dat wordt uitgevoerd, dan heeft __name__ de waarde "__main__", anders is de naam van de module als string.
- if __name__=='__main__':
  …

Testen



Testen

- Nose is module waarmee men programma's kan testen.
- Elke test module (naam moet beginnen met test_) moet bevatten:
  - import van nose en de module die getest moet worden.
  - Functies die de module testen (naam moet beginnen met test_).
  - Een functie call om de test functies uit te voeren.
- Drie mogelijkheden:
  - Pass of . : de eigenlijke waarde is de verwachte waarde.
  - Fail of F : de eigenlijke waarde is verschillend van de verwachte waarde.
  - Error : er zit een fout in de test zelf.

test_temperature.py

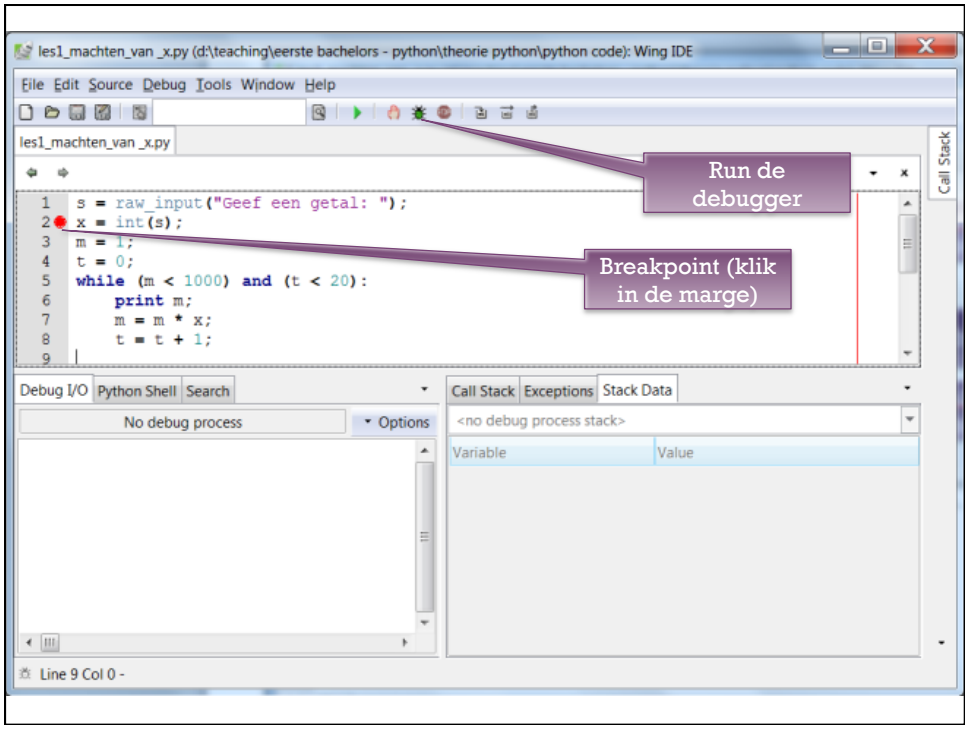

```
import nose
from temperature import above_freezing
def test_above_freezing():
    '''Test function for above_freezing.'''
    assert above_freezing(89.4), 'A temperature above freezing.'
    assert not above_freezing(-42), 'A temperature below freezing.'
    assert not above_freezing(0), 'A temperature at freezing.'

if __name__ == '__main__':
    nose.runmodule()
```

assert statement

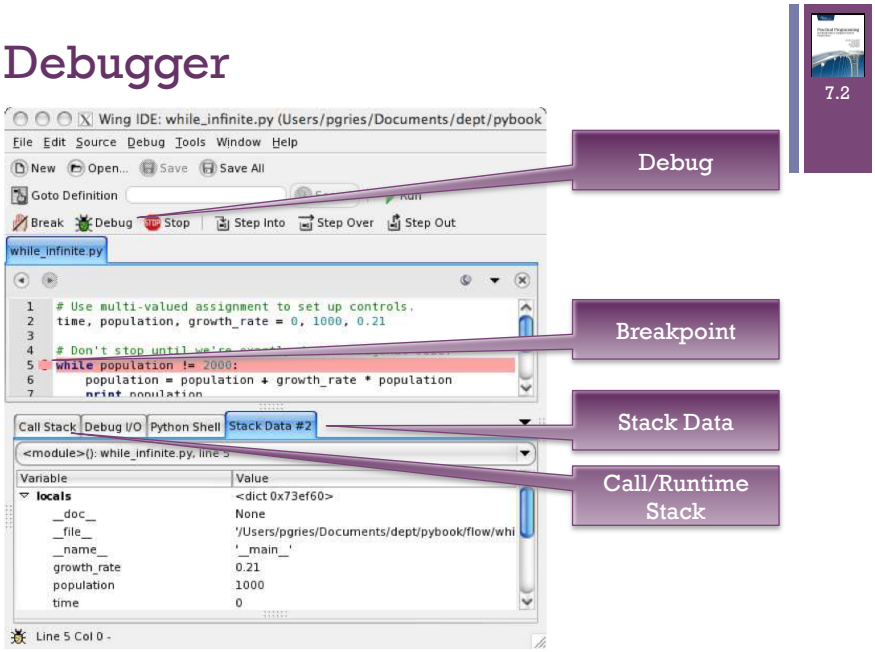
Boundary case

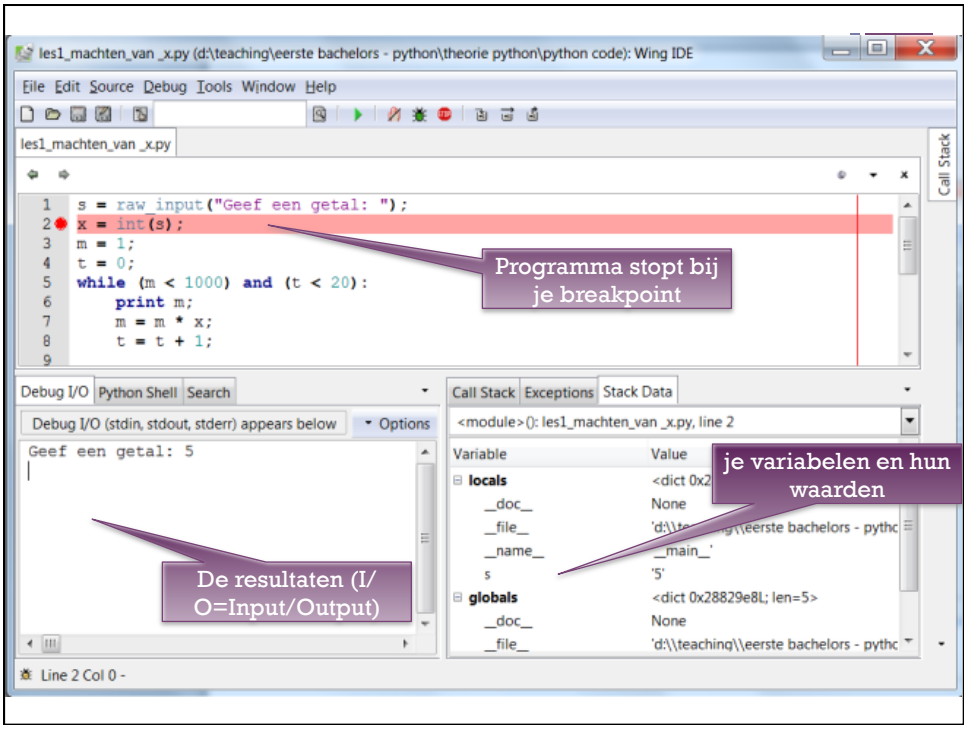
Stop en Sla over

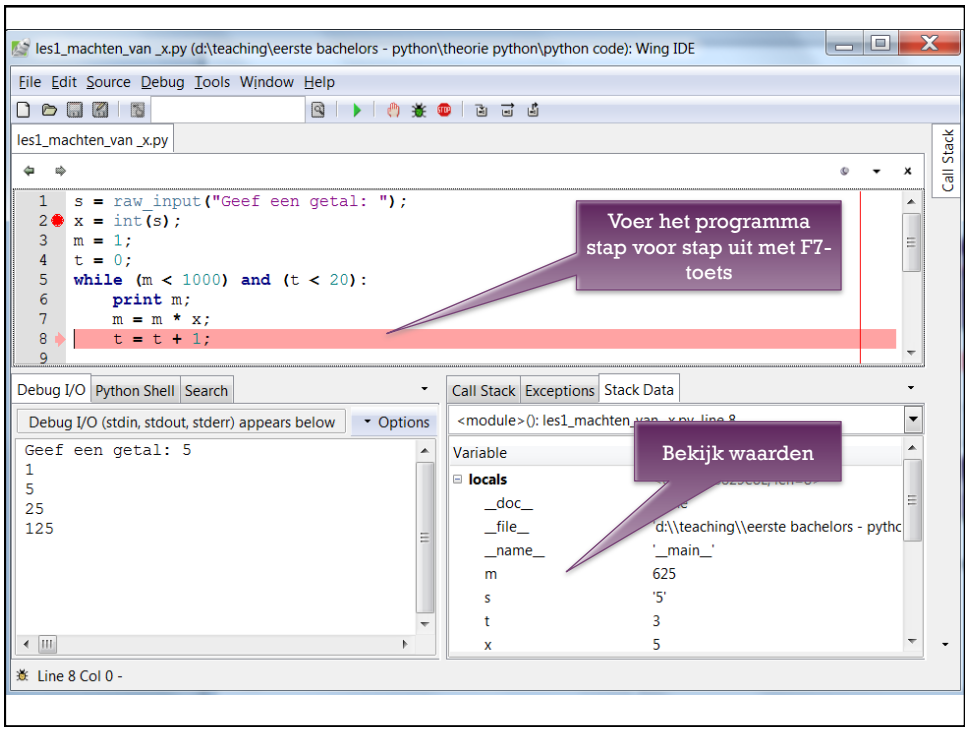
- break statement
- continue statement



Debugger


7.2
Wing IDE: while_infinite.py (Users/pgries/Documents/dept/pybook
File Edit Source Debug Tools Window Help
New Open... Save Save All
Goto Definition
Break Debug Stop Step Into Step Over Step Out
while_infinite.py
Use multi-valued assignment to set up controls.
time, population, growth_rate = 0, 1000, 0.21
while population != 2000:
population = population + growth_rate * population
Call Stack Debug I/O Python Shell Stack Data #2
<module>(): while_infinite.py, line 5
Variable Value
locals <dict 0x73ef60>
__doc__ None
__file__ '/Users/pgries/Documents/dept/pybook/flow/whi
__name__ '__main__'
growth_rate 0.21
population 1000
time 0
Line 5 Col 0 -
Debug
Breakpoint
Stack Data
Call/Runtime Stack


les1_machten_van _x.py (d:\teaching\eerste bachelors - python\theorie python\python code): Wing IDE
File Edit Source Debug Tools Window Help
les1_machten_van _x.py
s = raw_input("Geef een getal: ");
x = int(s);
m = 1;
t = 0;
while (m < 1000) and (t < 20):
print m;
m = m * x;
t = t + 1;
Call Stack
Programma stopt bij je breakpoint
Debug I/O Python Shell Search
Debug I/O (stdin, stdout, stderr) appears below
Options
Geef een getal: 5
Call Stack Exceptions Stack Data
<module>(): les1_machten_van _x.py, line 2
Variable Value
locals
__doc__ None
__file__
__name__ __main__'
s '5'
globals <dict 0x28829e8L; len=5>
__doc__ None
__file__
je variabelen en hun waarden
De resultaten (I/O=Input/Output)
Line 2 Col 0 -



+ Command-Line Arguments

- Wanneer we een programma (.py file) runnen, kunnen we argumenten meegeven, zoals bij een function call of methode.
- De waarden van de argumenten worden opgeslagen in de variabele argv die een lijst van strings is binnen de systeem module sys.
- sys.argv[0] : bevat altijd de naam van het Python programma dat gerund wordt.
- De rest van de command-line argumenten worden opgeslagen in sys.argv[1], sys.argv[2], …

Command-Line Arguments


5.10

```
''' Display the lines of data.txt from the given starting line number to the
given end line number.

Usage: read_lines_range.py start_line end_line '''

import sys

if __name__ == '__main__':

    # get the start and end line numbers
    start_line = int(sys.argv[1])
    end_line = int(sys.argv[2])

    # read the lines of the file and store them in a list
    data = open('data.txt', 'r')
    data_list = data.readlines()
    data.close()

    # display lines within start to end range
    for line in data_list[start_line:end_line]:
        print line.strip()
```

Docstring

Commentaar

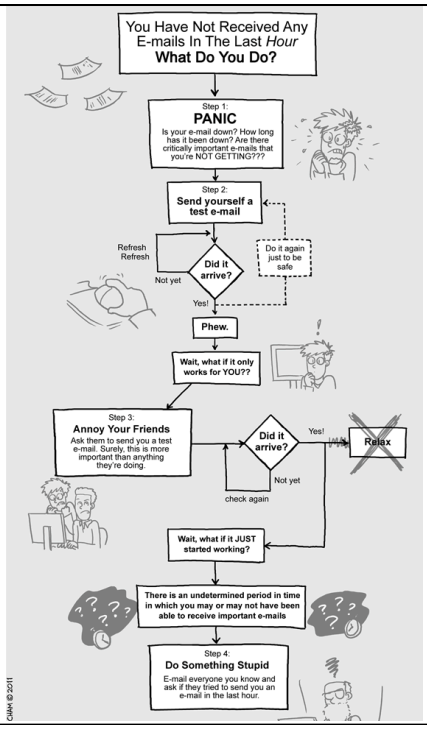
Maak …


7.4

geen te grote nest ☺

Informatica

Les 4



Keuzes maken: if statement

if statement

Een *if statement* heeft de vorm

```
if condition:
        if-block
else:
        else-block
```

clause

- condition : een expressie die True of False is
- if-block/else-block : een lijst van Python statements (zoals de *block* bij functies, enkel *return* is niet toegelaten)

Iteraties: for loop statement

Counted loop

for statement

5.4
7.1

Een *for loop statement* heeft de vorm

```
for variable in seq:
    block
```

- variable : een variabele
- seq : een lijst, tupel of string
- block : een lijst van Python statements
  (zoals de *block* bij functies, enkel *return* is niet toegelaten)

Iteraties: while loop statement

Conditional loop

while loop statement

Een *while (of conditional) loop statement* heeft de vorm

```
while condition:
    block
```

- condition : een expressie die True of False is
- block : een lijst van Python statements
  (zoals de *block* bij functies, enkel *return* is niet toegelaten)

Recursie

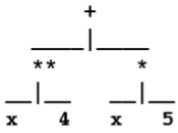
Recursie

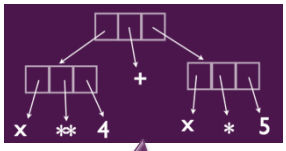
Hier zit geen limiet op, maar in praktijk stopt het ergens

- Recursie = zelf-referentie
  oproepen van een functie in de definitie van de functie zelf
  - Data-structuren
    - Expressions waarvan de operandi ook expressions zijn
    - Geneste lijsten
    - …
  - Functies
    - Faculteit functie !
    - Exponent functie
    - Afleiden
    - Fibonacci getallen
    - …
- Het gedrag kan voorgesteld worden door een boom. Het aantal vertakkingen kan lineair, logaritmisch of exponentieel toenemen.

Voorbeeld: Recursieve Datastructurem

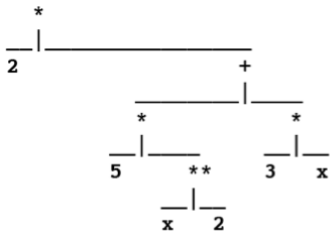
- x^4+5x





Kan je realizeren met geneste tupels!

- $2(5x^2+3x)$



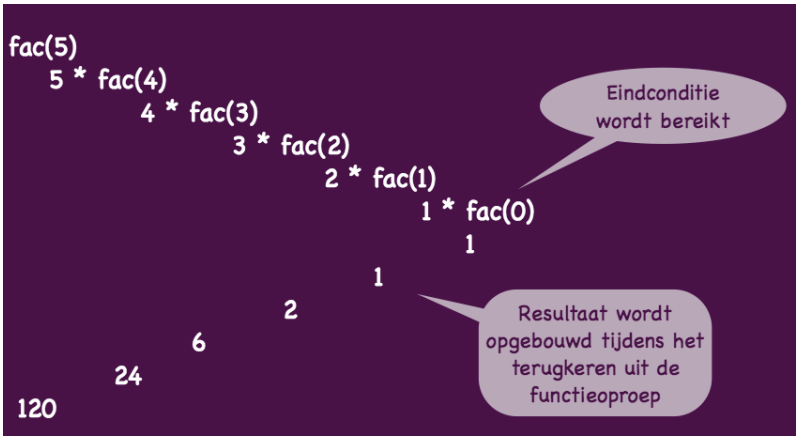
Voorbeeld: Recursieve Functies

- $$n! = \begin{cases} 1 & n = 0 \\ n*(n-1)! & n > 0 \end{cases}$$

- def fac(n):
  if n==0:
  return 1
  else:
  return n*fac(n-1)

- >>> fac(5)

Voorbeeld: Recursieve Functies



Voorbeeld: Recursieve Functies

- def power(a, n):
  if n==0:
  return 1
  else:
  return a*power(a, n-1)

$$a^n = \begin{cases} 1 & n = 0 \\ a * a^{n-1} & n > 0 \end{cases}$$

- def fast_power(a, n):
  if n==0:
  return 1
  elif n%2==0:
  return fast_power(a, n/2)**2
  else:
  return a*fast_power(a, n-1)

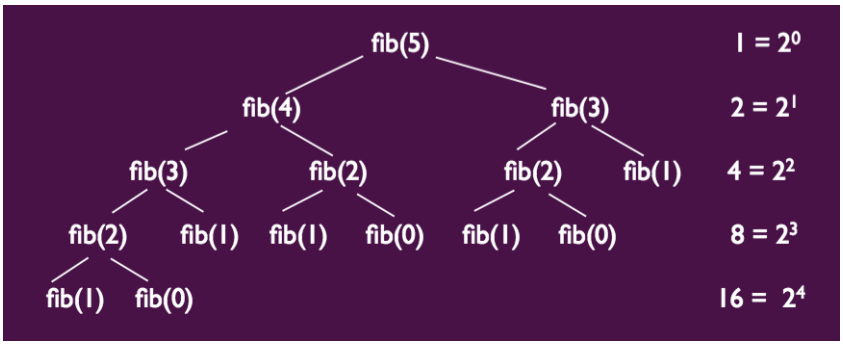
$$a^n = \begin{cases} 1 & n = 0 \\ (a^{n/2})^2 & n \ \ even \\ a * a^{n-1} & anders \end{cases}$$

Voorbeeld: Recursieve Functies

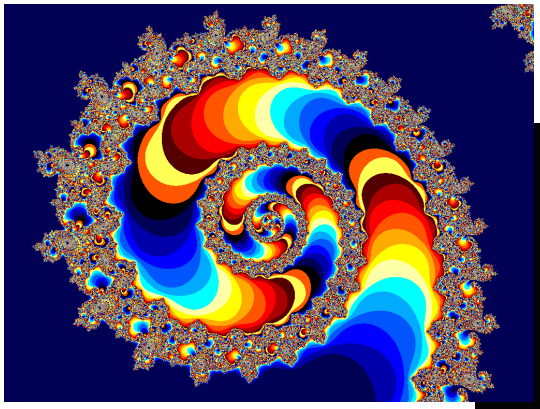
- $$fib(n) = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ fib(n-1) + fib(n-2) & anders \end{cases}$$

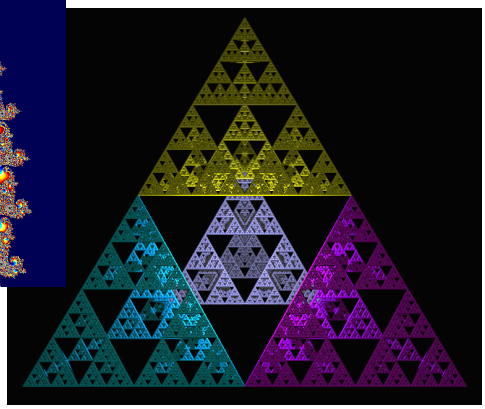
- def fib(n):
  if n==0:
  return 0
  elif n==1:
  return 1
  else:
  return fib(n-1)+fib(n-2)

Voorbeeld: Recursieve Functies



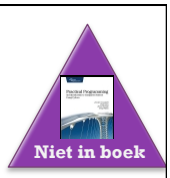
Fractalen: weerkerende patronen op elk niveau van detail





www.bugman123.com/Fractals/

Recursie: conclusies



- Recursie = oproepen van functie in functie zelf
- Elke aanroep moet bij uitvoering verschillend zijn
- Er 'moet' een stopconditie zijn
- Elke aanroep blijft in het geheugen met zijn lokale variabelen
- Er is staart- en boomrecursie

Files

Een eerste kennismaking met algoritmes …

Bestanden / Files

5.10

- Een *file* is een geordende sequentie van bytes en dient om data bij te houden. Het *formaat* (extensie) bepaalt hoe de bytes moeten georganiseerd en geïnterpreteerd worden.
- Files kunnen worden geopend om
  - Lezen / read : 'r'
  - Schrijven / write : 'w'
  - Toevoegen / append : 'a'

```
>>> file = open('naam bestand', 'r')
```

of 'w' of 'a' (modes)

Parsen

- Het resultaat van de built-in functie open is niet de inhoud van de file, maar een file object wiens methodes toegang verlenen tot de inhoud die bewaard wordt als een lijst van strings.
- Een tekst file is een sequentie van *lijnen* (*lines*) waarbij elke lijn een string is incl. het end-of-line teken \n en kan gelezen worden met de readline methode.

Voorbeeld: txt file

- Doordat open de inhoud van een file voorstelt als een lijst van strings, kunnen we het file object in een for loop gebruiken. Hierbij wordt readline automatisch aangeroepen.
- planeten.txt:

```
Mercury
Venus
Earth
Mars
```

```
>>> planeten=open('planeten.txt', 'r')
>>> for line in planeten:
        print len(line.strip())
```

Nadien best planeten.close()

- string.strip() geeft een kopij van string waar de spaties, tabs en newlines aan het begin en einde zijn 'afgestript'.

Voorbeeld: txt file

- Als je de getalwaarde nodig hebt van een string, moet je een conversie doen naar int of float.
- numbers.txt

```
12
13.5
0.2
6
```

-
```
numbers=open('numbers.txt', 'r')
sum=0
for n in numbers:
        sum=sum + float(n)
print sum
```

Eén "record" per lijn


```
1  Punten Informatica
2  #Ann Dooms
3  #January 2012
4  13
5  10
6  6
7  18
8  12
9  13
10 14
11 8
```

text_parse.py

```
1  def copy_on_screen(filename):
2      input_file = open(filename, 'r')
3      for line in input_file:
4          line = line.strip()
5          print line
6      input_file.close()
7
8
```

Titel en commentaar overslaan


```
def skip_header(rfile):
    line = rfile.readline()
    line = rfile.readline()
    while line.startswith ('#'):
        line = rfile.readline()
    return line

def copy_on_screen_alt(filename):
    input_file = open(filename, 'r')
    line = skip_header(input_file).strip()
    print line
    for line in input_file:
        line = line.strip()
        print line
    input_file.close()
```

Lees titel

Gooi titel weg & lees volgende lijn

Zolang als lijn start met commentaar, lees volgende lijn

Eerste echte data lijn

Naar files schrijven


```
>>> myfile = open('planeten.txt', 'a')
>>> myfile.write('Moon')
>>> myfile.close()
```

```
Mercury
Venus
Earth
Mars
Moon
```

Alles samen

Files uitlezen en wegschrijven

```
def process_student_grades(infilename, outfilename):
    input_file = open(infilename, 'r')
    output_file = open(outfilename, 'w')
    output_file.write(input_file.readline())
    for line in input_file:
        if not line.startswith('#'):
            vn, fn, th, oef = line.split()
            avg = (float(th)+float(oef))/2
            new_line = vn + ' ' + fn + ' ' + str(avg) + '\n'
            output_file.write(new_line)
    input_file.close()
    output_file.close()
    return None
```

Meerdere velden in record

Informatica

Les 5

Algoritmes

Algoritmes

Een algoritme is een lijst van stappen om een taak uit te voeren.

- Top-down design: een probleem verder opdelen in deelproblemen tot die kunnen vertaald worden naar (Python) code.
- Documenteren
- Testen
- Debuggen
- Performantie
  - Nagaan door middel van execution profiling, i.e. experimenteel meten hoe lang het duurt om het programma te runnen (*time*) en hoeveel geheugen het gebruikt (*space*).
  - *Worst-case* (slechtste geval) afschatting maken van de grootte-orde van het aantal "stappen" in functie van de grootte van de input.
- Afweging tussen "eenvoudige" code en performantie.

Algoritmes

We hebben reeds enkele algoritmes gezien in "simpele" functies/ modules die onder andere gebruik maken van

- Keuzes maken
- Iteraties
- Recursie

Nu gaan we een stapje verder en bestuderen algoritmes om twee belangrijke opdrachten uit te voeren die we bijna continu aan onze computer vragen:

- Zoeken
- Sorteren

Zoeken / Search

Zoek index van het kleinste getal


```
>>> counts = [809, 834, 477, 478, 307, 122, 96, 102, 324, 476]
>>> counts.index(min(counts))
6
```

Zoek index van de twee kleinste getallen

- Oplossing 1: zoek index van kleinste getal op bovenstaande wijze, verwijder, zoek opnieuw index van kleinste getal;
- Oplossing 2: sorteer, neem de twee kleinste en zoek hun indices;
- Oplossing 3: loop éénmaal door de lijst en hou ondertussen de indices van de twee kleinste gevonden getallen bij en update indien nodig.

Oplossing 1: Find, remove, find

Top-down aanpak

```
def find_two_smallest(L):
    '''Return a tuple of the indices of the two smallest values in list L.'''

    find the index of the minimum element in L
    remove that element from the list
    find the index of the new minimum element in the list
    return the two indices
```

In help van list vinden we geen methode die dit doet

Oplossing 1: Find, remove, find

Verfijnd

```
def find_two_smallest(L):
    '''Return a tuple of the indices of the two smallest values in list L.'''

    get the minimum element in L
    find the index of that minimum element
    remove that element from the list
    find the index of the new minimum element in the list
    return the two indices
```

Opnieuw min en list.index gebruiken

list.remove

Oplossing 1: Find, remove, find

Als len(L)=n, dan doet dit algoritme in het slechtste geval 4n stappen.

```
def find_two_smallest(L):
    '''Return a tuple of the indices of the two smallest values in list L.'''

    smallest = min(L)
    min1 = L.index(smallest)
    L.remove(smallest)
    next_smallest = min(L)
    min2 = L.index(next_smallest)

    L.insert(min1, smallest)
    if min1 <= min2:
        min2 += 1

    return (min1, min2)
```

L terug "repareren"

Oplossing 2: Sort, identify, index


```
def find_two_smallest(L):
    '''Return a tuple of the indices of the two smallest values in list L.'''

    sort a copy of L
    get the two smallest numbers
    find their indices in the original list L
    return the two indices
```

In help van list vinden we de list.sort methode. Deze overschrijft L, dus moeten we de originele lijst kopiëren!

Oplossing 2: Sort, identify, index

Als len(L)=n, neemt dit n stappen.

```
def find_two_smallest(L):
    '''Return a tuple of the indices of the two smallest values in list L.'''

    temp_list = L[:]
    temp_list.sort()
    smallest = temp_list[0]
    next_smallest = temp_list[1]
    min1 = L.index(smallest)
    min2 = L.index(next_smallest)
    return (min1, min2)
```

Straks zien we dat dit in het slechtste geval nlog(n) stappen vraagt.

2n stappen in het slechtste geval

Oplossing 3: Walk through list


```
def find_two_smallest(L):
    '''Return a tuple of the indices of the two smallest values in list L.'''

    examine each value in the list in order
    keep track of the indices of the two smallest values found so far
    update these values when a new smaller value is found
    return the two indices

def find_two_smallest(L):
    '''Return a tuple of the indices of the two smallest values in list L.'''

    set min1 and min2 to the indices of the smallest and next-smallest
    values at the beginning of L

    examine each value in the list in order
        update these values when a new smaller value is found
    return the two indices
```

for loop over de indices van L

Oplossing 3: Walk through list

Top-down aanpak

```
def find_two_smallest(L):
    '''Return a tuple of the indices of the two smallest values in list L.'''

    # set min1 and min2 to the indices of the smallest and next-smallest
    # values at the beginning of L
    if L[0] < L[1]:
        min1, min2 = 0, 1
    else:
        min2, min1 = 1, 0

    # examine each value in the list in order
    for i in range(2, len(L)):

        L[i] is larger than both min1 and min2, smaller than both, or
        in between.
        if L[i] is larger than both min1 and min2, skip it
        if L[i] is smaller than min1 and min2, update them both
        if L[i] is in between, update min2

    return (min1, min2)
```

Oplossing 3: Walk through list


```
def find_two_smallest(L):
    '''Return a tuple of the indices of the two smallest values in list L.'''

    # set min1 and min2 to the indices of the smallest and next-smallest
    # values at the beginning of L
    if L[0] < L[1]:
        min1, min2 = 0, 1
    else:
        min2, min1 = 1, 0

    # examine each value in the list in order
    for i in range(2, len(L)):

        # L[i] is larger than both min1 and min2, smaller than both, or
        # in between.

        # New smallest?
        if L[i] < L[min1]:
            min2 = min1
            min1 = i

        # New second smallest?
        elif L[i] < L[min2]:
            min2 = i

    return (min1, min2)
```

Als len(L)=n, doet dit algoritme slechts n stappen

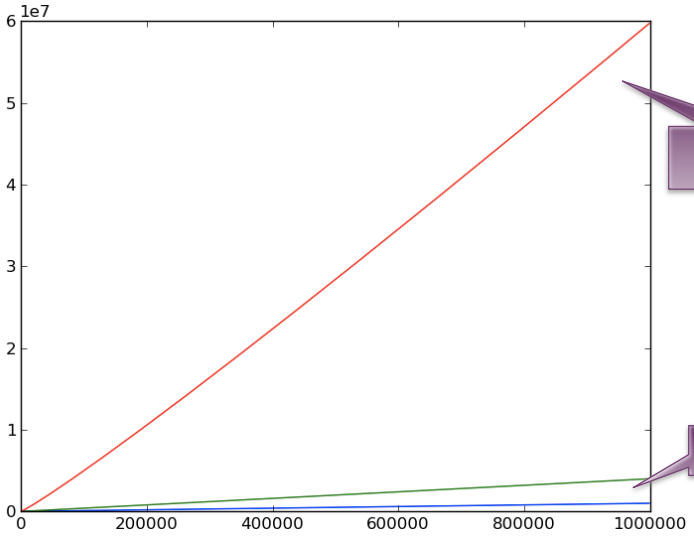
Timing van de 3 oplossingen

Time voor len(L)=n=1400

Algorithm	Running Time (ms)
Find, remove, find	1.117
Sort, identify, index	2.128
Walk through the list	1.472

Het eerste algoritme is het eenvoudigste én het snelste, maar wat als n zeer groot kan zijn?

Performantie van de 3 oplossingen



3n+ nlog(n)

n en 4n

Soorten zoek-algoritmes

- Linear search
  - Basic linear search
  - For loop versie van linear search
  - Sentinel Search
- Binary search
- …

Waar zit de methode list.index?

Basic Linear Search


```
def linear_search(v, L):
    '''Return the index of the first occurrence of v in list L, or return len(L)
    if v is not in L.'''

    i = 0
    # Keep going until we reach the end of L or until we find v.
    while i != len(L) and L[i] != v:
        i = i + 1

    return i
```

Deze test kunnen we vermijden via een for loop.

For loop versie van Linear Search


```
def linear_search(v, L):
    '''Return the index of the first occurrence of v in list L, or return len(L)
    if v is not in L.'''

    i = 0
    for value in L:
        if value == v:
            return i

        i += 1

    return len(L)
```

Sentinel Search


```
def linear_search(v, L):
    '''Return the index of the first occurrence of v in list L, or return len(L)
    if v is not in L.'''

    # Add the sentinel.
    L.append(v)

    i = 0

    # Keep going until we find v.
    while L[i] != v:
        i = i + 1

    # Remove the sentinel.
    L.pop()

    return i
```

Vuil truukje!

Timing van Linear Search

Waar het te zoeken item zit:

	Case	basic	for	sentinel	list.index
Beste geval:	First	0.01	0.01	0.03	0.01
Gemiddeld geval:	Middle	138	69	62	17
Slechtste geval:	Last	273	139	124	35

n=1.000.000

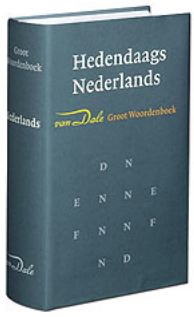
Figure 11.1: Running times for linear search, milliseconds

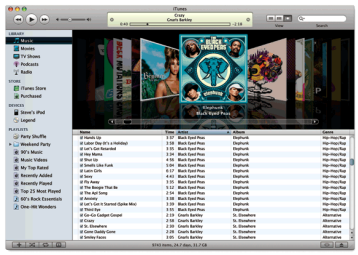
- 7 keer langer dan list.index
- Tijd verdubbelt als input verdubbelt.

- 4 keer langer dan list.index
- Tijd verdubbelt als input verdubbelt.

Binary Search

Wat als lijst gesorteerd is?





Binary Search


```
def binary_search(v, L):
    """Return the index of the leftmost occurrence of v in list L, or -1 if
    v is not in L."""
    # Mark the left and right indices of the unknown section.
    i = 0
    j = len(L) - 1

    while i != j + 1:
        m = (i + j) / 2
        if L[m] < v:
            i = m + 1
        else:
            j = m - 1

    if 0 <= i < len(L) and L[i] == v:
        return i
    else:
        return -1
```

Timing van Binary Search

N	Steps Required
100	7
1000	10
1,0000	14
10,0000	17
100,000	24
1,000,000	27

Figure 11.2: Logarithmic growth

n=100.000

Case	list.index	binary_search	Ratio
First	0.03	0.05	0.66
Middle	107	0.04	2643
Last	15.73	0.04 (Wow!)	4304

Figure 11.3: Running times for binary search

In elke stap maken we de keuze tussen twee waarden. Omgekeerd krijgen we dus een logaritmisch gedrag voor het aantal stappen in functie van het aantal waarden.

Het maakt niet uit waar het item zit!

Conclusie

- Linear search is de eenvoudigste methode om een waarde in een lijst te vinden.
  Gemiddeld gezien is de benodigde tijd proportioneel met de lengte van de lijst.
- Binary search is complexer, maar veel vlugger.
  Gemiddeld gezien is de benodigde tijd proportioneel met de logaritme van de lengte van de lijst, MAAR werkt alleen als de lijst gesorteerd is.

Sorteren / Sort

Soorten sorteer-algoritmes

- Selection sort
- Insertion sort
- Merge sort
- Bubble sort (zie oefeningen 11.7.3-6 in boek)
- Quicksort (niet voor nu)
- Heapsort (niet voor nu)
- …

Waar zit de methode list.sort?

Selection Sort

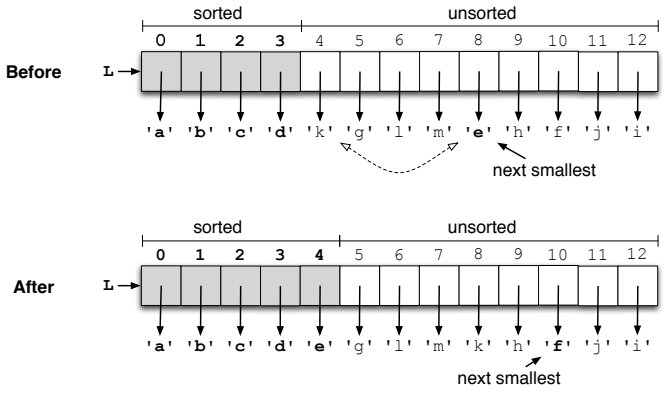


Figure 11.5: Selection sort

Selection Sort


```
def find_min(L, b):
    smallest = b
    i=b+1
    while i != len(L):
        if L[i] < L[smallest]:
            smallest = i
        i=i+1
    return smallest

def selection_sort(L):
    i=0
    while i != len(L):
        smallest = find_min(L, i)
        L[i], L[smallest] = L[smallest], L[i]
        i=i+1
```

In totaal $n + (n-1) + (n-2) + \ldots + 1 = \frac{n(n+1)}{2}$

Insertion Sort


```
def insert(L, b):
    i=b
    while i != 0 and L[i - 1] >= L[b]:
        i=i-1
    value = L[b]
    del L[b]
    L.insert(i, value)

def insertion_sort(L):
    i=0
    while i != len(L):
        insert(L, i)
        i=i+1
```

In het ergste geval: $n + (n-1) + (n-2) + \ldots + 1 = \frac{n(n+1)}{2}$

Timing van Selection & Insertion Sort

List Length	Selection Sort	Insertion Sort	list.sort
10	0.1	0.1	0.0
1000	481	300	0.1
2000	1640	1223	0.2
3000	3612	2772	0.4
4000	6536	4957	0.6
5000	10112	7736	0.6
10000	40763	32382	1.3

Figure 11.6: Running times for selection and insertion sort, milliseconds

Input lengte n verdubbelen, vermenigvuldigt de running time met 4!

Wow, wegens gebaseerd op ideeën uit binary search!

Merge Sort

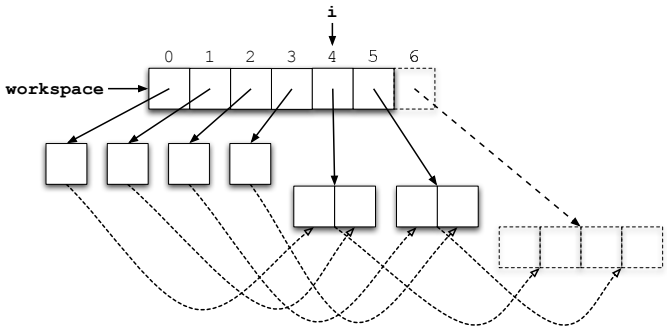


Figure 11.8: List of lists in mergesort

Merge Sort


```
def merge(L1, L2):
    newL = []
    i1 = 0
    i2 = 0
    while i1 != len(L1) and i2 != len(L2):
        if L1[i1] <= L2[i2]:
            newL.append(L1[i1])
            i1 += 1
        else:
            newL.append(L2[i2])
            i2 += 1
    newL.extend(L1[i1:])
    newL.extend(L2[i2:])
    return newL
```

2 gesorteerde lijsten samenritsen

overschot overhevelen

```
def mergesort(L):
    workspace = []
    for i in range(len(L)):
        workspace.append([L[i]])
    i=0
    while i < len(workspace) - 1:
        L1 = workspace[i]
        L2 = workspace[i + 1]
        newL = merge(L1, L2)
        workspace.append(newL)
        i += 2
    if len(workspace) != 0:
        L[:] = workspace[-1][:]
```

lijst van lijstjes

Merge Sort

Het maken van de lijstjes kost al O(n) werk. Het ritsen?

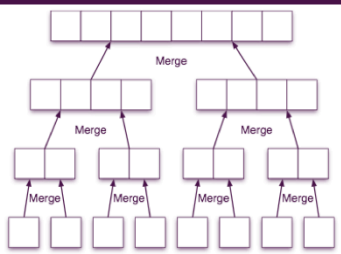


Figure 11.9: Steps in mergesort

$n/2^i$ koppels ritsen van 2^{i-1}

. . .

n/8 koppels ritsen van 4

n/4 koppels ritsen van 2

n/2 koppels ritsen van 1

Op elk niveau: $n/2^i$ keer 2^i stapjes. Dus n stapjes.

Men kan een lijst $\log_2(n)$ keer in 2 delen. Dus zijn er $\log_2(n)$ niveaus.

Conclusie

- Selection en Insertion sort zijn eenvoudige methodes om te sorteren. Gemiddeld gezien is de benodigde tijd proportioneel met het kwadraat van de lengte van de lijst n.
- Merge sort is complexer, maar vlugger. Gemiddeld gezien is de benodigde tijd proportioneel met nlog(n).

Big-Oh

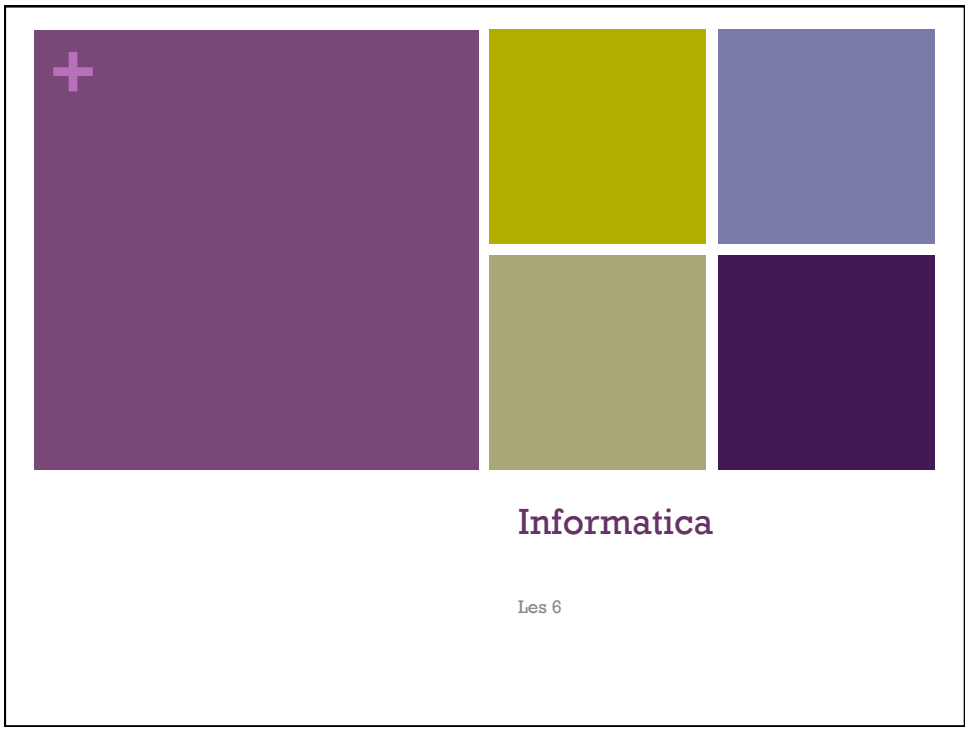
Grootte-ordes op een rijtje

Als *f* en *g* functies zijn, zodat voor voldoende grote *x*, *f(x)* begrensd is door α *g(x)*, met α een constante, dan zeggen we dat

$$f(x)=O(g(x)).$$

$$1 < \log(n) < \sqrt{n} < n < n.\log(n) < n^{k-1} < n^k < 2^n < n!$$

- fast_power, Binary search zijn O(log(n))
- fac(n), power(n), zoek-oplossing 1en 2, Linear search zijn O(n)
- Zoek-oplossing 3 en Merge sort zijn O(nlog(n))
- Selection en Insertion sort zijn $O(n^2)$
- De recursieve versie vam fib(n) is $\pm O(2^n)$


Informatica
Les 6

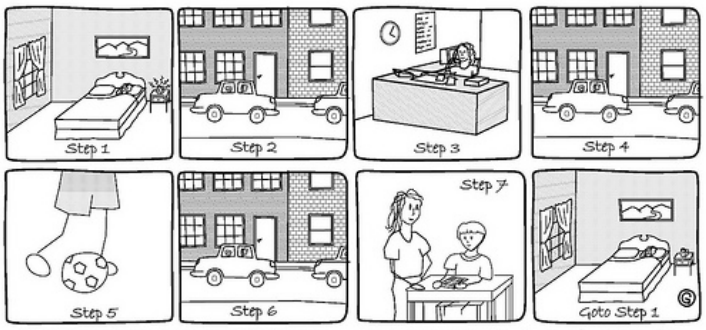

Python Wrap Up

Python Wrap Up

Van variabelen …

"Just a darn minute! — Yesterday you said that X equals **two**!"

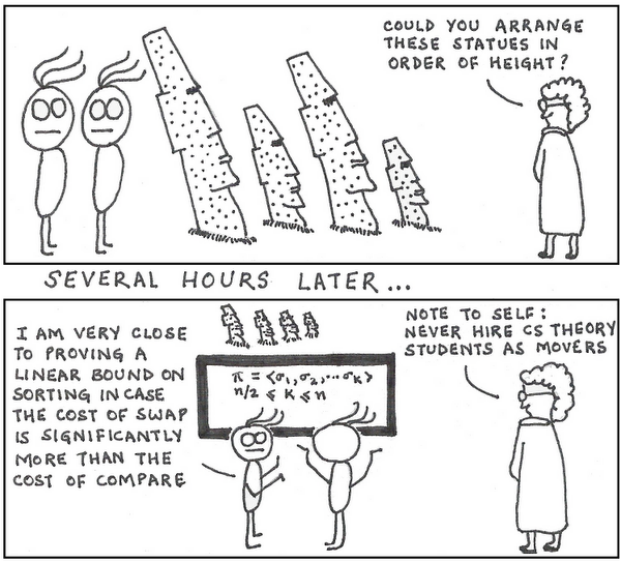
Python Wrap Up



Ever have the sneaking suspicion you are trapped in an infinite loop?

… over loops …

Python Wrap Up



... tot algoritmes!

Python Basics

Python Statements

In shell of in .py files voor herbruikbaarheid en/of lange algoritmes.

- Expressions/types & operatoren
  - immutable versus mutable
- Assignments/variabelen
  - lokale versus globale
- Functies:
  - Definitie met eventuele parameters
  - Call met eventuele argumenten
  - Laat recursie toe
- 'Gebruiksvriendelijke' statements:
  - return
  - import
  - print
  - Modules maken ons het leven makkelijk!
- Control flow statements:
  - if
  - for
  - while

Objecten en Methodes

4.3

- Een waarde in Python (getallen, strings, tupels, maar ook functies, matrices, ...) noemt men een *object*. Het heeft een bepaald type en een plaats in het geheugen.
- Objecten bevatten *methodes*, i.e. speciale type-afhankelijke functies die in objecten zitten en die je aanroept met een "." (je kan hiermee ook ketens maken).

```
object.method_naam(parameterlist)
```

- Operatoren zijn eigenlijk mooie syntactische voorstellingen voor methodes. De read-fase doet het "vuile werk".

Opmerking

Na het importeren van een module, nemen we de functies en variabelen hierbinnen ook vast met een "."

Algoritmes

Algoritmes

Een algoritme is een lijst van stappen om een taak uit te voeren.

- Top-down design: een probleem verder opdelen in deelproblemen tot die kunnen vertaald worden naar (Python) code.
- Documenteren
- Testen
- Debuggen
- Performantie
  - Nagaan door middel van execution profiling, i.e. experimenteel meten hoe lang het duurt om het programma te runnen (*time*) en hoeveel geheugen het gebruikt (*space*).
  - *Worst-case* (slechtste geval) afschatting maken van de grootte-orde van het aantal "stappen" in functie van de grootte van de input.
- Afweging tussen "eenvoudige" code en performantie.

Toepassingen

- Werken met tekst bestanden (*text files*)
- Zoeken
  - Linear search
    - Basic linear search
    - For loop versie van linear search
    - Sentinel Search
  - Binary search
- Sorteren
  - Selection sort
  - Insertion sort
  - Merge sort

Een laatste algoritme ontwikkelen van naaldje tot draadje …

… totdat het geen geheimen meer heeft …

… of toch?

Data Hiding/Watermerken

Zie je een verschil?

Data Hiding

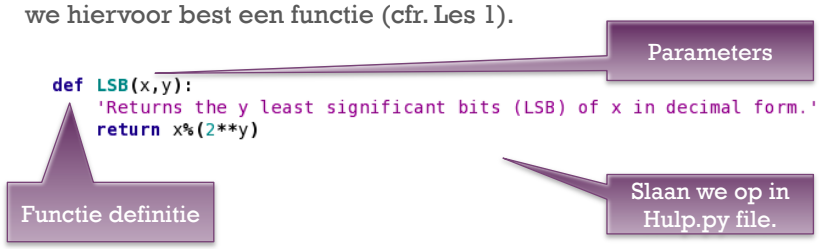
Een twee-dimensionaal raster van pixels.

- Een digitaal kleurenbeeld heeft 3 kleurcomponenten per pixel: R, G en B.
- Elke component heeft een waarde tussen 0 en 255.
  - Wit = (255, 255, 255)
  - = (121, 72, 120)
  - Zwart = (0,0,0)
- De 256 mogelijke waarden kunnen voorgesteld worden door 8 bits.
  - 2 = 00000010
  - 65 = 01000001
  - 255 = 11111111
- De invloed van de laatste (i.e. rechtse) bits op de kleur is miniem.
  - Laatste bit wijzigen: ±1
  - Twee laatste bits wijzigen: ±1,2,3

Noemen we de *least significant bits.*

Data Hiding

- Laten we de (twee) laatste bits van beide beelden vergelijken.
- Probleem opdelen in stappen:
  1. Laatste bits van een 8-bit getal in decimale vorm bepalen.
  2. Toepassen op de drie componenten van een beeld.
- Aangezien we Stap 1 drie keer willen toepassen in Stap 2, schrijven we hiervoor best een functie (cfr. Les 1).



Data Hiding

- Voor Stap 2 hebben we de Python Imaging Library (PIL) nodig.
- We vinden alle info over de Image module (cfr. Les 2) op http://www.pythonware.com/library/pil/handbook/image.htm
- We creëren de file LSB-SimpleDetect.py.
- De Image en onze Hulp modules moeten geïmporteerd worden om een beeld te kunnen openen.
- Door het algoritme interactief te maken, kunnen we de gebruiker makkelijk laten kiezen op welke bestanden hij het toepast.

Data Hiding

Docstring

```
'Detecteert een watermerk in de y LSB van een RGB beeld.'

import Image
#Methodes genomen van http://www.pythonware.com/library/pil/handbook/image.htm
from Hulp import LSB

name_wm=raw_input('Geef de naam van de gewatermerkte foto: ')
im_wm=Image.open(name_wm)
im_wm.show()
```

Commentaar

We hebben enkel LSB nodig.

Hoe kunnen we nu de pixels uit een beeld opvragen?

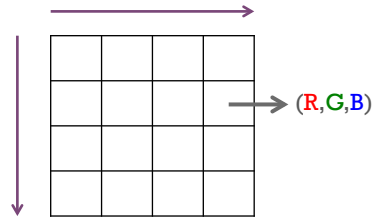
(New in 1.1.6) In 1.1.6 and later, **load** returns a pixel access object that can be used to read and modify pixels. The access object behaves like a 2-dimensional array, so you can do:

```
pix = im.load()
print pix[x, y]
pix[x, y] = value
```

Zie PIL website

Data Hiding

- Nu moeten we alle pixels uit het gegeven beeld opvragen.



- Twee for loops (cfr. Les 3) over de lengte en breedte van het beeld.

size

im.size => (width, height)

Image size, in pixels. The size is given as a 2-tuple (width, height).

Zie PIL website

Data Hiding

```
'Detecteert een watermerk in de y LSB van een RGB beeld.'

import Image
#Methodes genomen van http://www.pythonware.com/library/pil/handbook/image.htm
from Hulp import LSB

name_wm=raw_input('Geef de naam van de gewatermerkte foto: ')
im_wm=Image.open(name_wm)
im_wm.show()
im_wm_pix=im_wm.load()

y=raw_input('Geef het aantal te gebruiken LSB: ')
y=int(y)

for i in range(im_wm.size[0]):
    for j in range(im_wm.size[1]):
        pixel_rood=LSB(im_wm_pix[i,j][0],y)
        pixel_groen=LSB(im_wm_pix[i,j][1],y)
        pixel_blauw=LSB(im_wm_pix[i,j][2],y)
        im_wm_pix[i,j]=(pixel_rood,pixel_groen,pixel_blauw)
```

Pixel object: twee-dimensionale lijst van (R,G,B) tupels

De gebruiker ook het aantal bits laten bepalen. De string moet omgezet worden in int!

Onze LSB functie aanroepen (*function call*) op elke kleurcomponent.

De pixel waarden overschrijven.

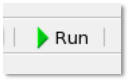
Data Hiding

Het resultaat slaan we op in een beeld, liefst met een naam gebaseerd op het originele beeld.

String methodes (cfr. Les 1)

```
name_wm=name_wm.strip('.png')+'_simple_watermark.png'
im_wm.save(name_wm)
wm=Image.open(name_wm)
wm.show()
```

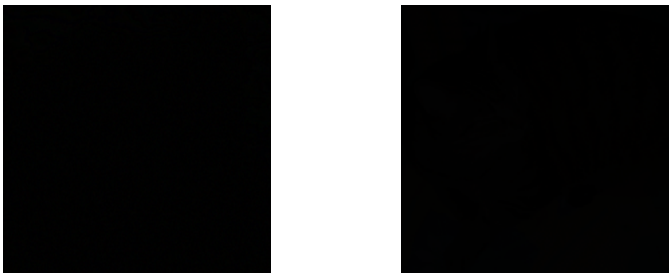
Nu runnen we de module op beide bomen in bloesem!



Data Hiding

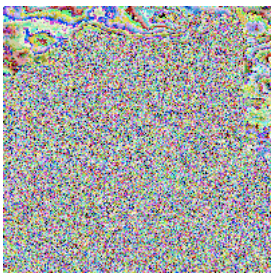
Geen verschil?

Data Hiding



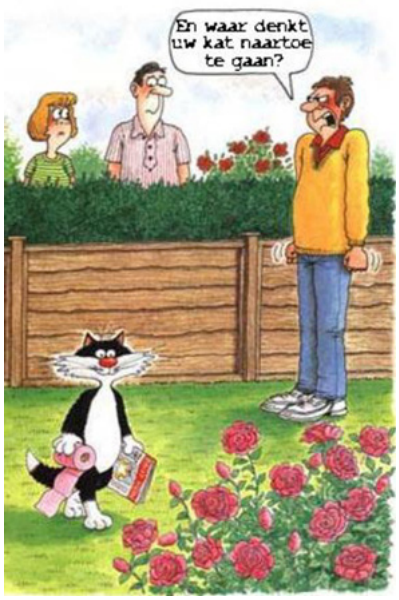
Laten we spelen met het contrast.

Data Hiding



We hebben de 'kat uit de boom gekeken' ☺ !

Data Hiding



Data Hiding

- Hoe kunnen we de kat in de boom krijgen?
- We moeten gewoon de laatste bits van de kleurcomponenten van het beeld wijzigen!

Uitbreiden Hulp.py

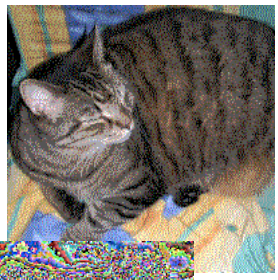
```
#x should be between 0 and 255

def LSB(x,y):
    'Returns the y least significant bits (LSB) of x in decimal form.'
    return x%(2**y)

def Replace_LSB(x,y,z):
    'Replaces the y LSB of x with those of z.'
    return x-LSB(x,y)+LSB(z,y)

def Contrast_LSB(x,y):
    'Maximizes the contrast of LSB(x,y) to be able to plot the values in the 256 colour range.'
    return int(LSB(x,y)*255./(2**y-1))
```

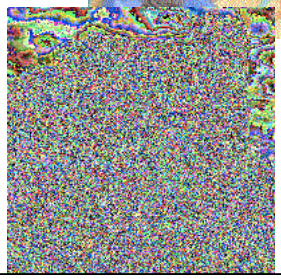
Vergeet het . niet (cfr. Les 1)!

```
'Plaatst een watermerk in de y LSB van een RGB beeld.'

import Image
#Methodes genomen van http://www.pythonware.com/library/pil/handbook/image.htm
from Hulp import Replace_LSB

name_orig=raw_input('Geef de naam van de originele foto: ')
im_orig=Image.open(name_orig)
im_orig.show()
im_orig_pix=im_orig.load()

name_wm=raw_input('Geef de naam van het watermerk: ')
wm=Image.open(name_wm)
wm.show()
wm_pix=wm.load()
y=raw_input('Geef het aantal te gebruiken LSB: ')
y=int(y)

if im_orig.size==wm.size:
    for i in range(im_orig.size[0]):
        for j in range(im_orig.size[1]):
            pixel_rood=Replace_LSB(im_orig_pix[i,j][0],y,wm_pix[i,j][0])
            pixel_groen=Replace_LSB(im_orig_pix[i,j][1],y,wm_pix[i,j][1])
            pixel_blauw=Replace_LSB(im_orig_pix[i,j][2],y,wm_pix[i,j][2])
            im_orig_pix[i,j]=(pixel_rood,pixel_groen,pixel_blauw)
else:
    print('De grootte van het watermerk komt niet overeen met deze van de originele foto.')

name_im_wm=name_orig.strip('.png')+'_watermarked.png'
im_orig.save(name_im_wm)
im_wm=Image.open(name_im_wm)
im_wm.show()
```

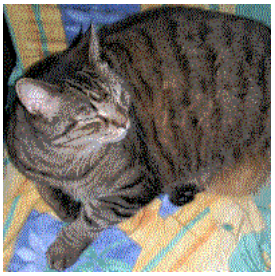
De if (cfr. Les 3) vangt hier de mogelijke mismatch tussen het originele beeld en het watermerk op.

```
'Detecteert een watermerk in de y LSB van een RGB beeld.'

import Image
#Methodes genomen van http://www.pythonware.com/library/pil/handbook/image.htm
from Hulp import Contrast_LSB

name_wm=raw_input('Geef de naam van de gewatermerkte foto: ')
im_wm=Image.open(name_wm)
im_wm.show()
im_wm_pix=im_wm.load()

y=raw_input('Geef het aantal te gebruiken LSB: ')
y=int(y)

for i in range(im_wm.size[0]):
    for j in range(im_wm.size[1]):
        pixel_rood=Contrast_LSB(im_wm_pix[i,j][0],y)
        pixel_groen=Contrast_LSB(im_wm_pix[i,j][1],y)
        pixel_blauw=Contrast_LSB(im_wm_pix[i,j][2],y)
        im_wm_pix[i,j]=(pixel_rood,pixel_groen,pixel_blauw)

name_wm=name_wm.strip('.png')+'_watermark.png'
im_wm.save(name_wm)
wm=Image.open(name_wm)
wm.show()
```





+ Geheimschrift 2.0



De kat zit verstopt in de twee laatste 'bitplanes' van de boom!

Ga en programmeer!

Veel succes!