BIO-INFORMATICA

OWE2A PYTHON II

PROGRAMMEREN IN PYTHON II

HAN_UNIVERSITY
OF APPLIED SCIENCES

CURRICULUM OVERZICHT

	Periode 1	Periode 2	Periode 3	Periode 4
1e jaar	Programmeren in Python (1)	Programmeren in Python (2)	Informatiesystemen met Python	Geautomatiseerde identificatie van eiwitten via sequentievergelijking
	Metabolisme	Opsporen van genetische mutaties bij erfelijke ziektes	Vergelijkende genoomanalyse: evolutie van virussen	Sequentie alignment (BLAST) en functionele eiwit analyse
2e jaar	Programmeren in Java	Datastructuren en algoritmen in Java	Analyse en ontwerptechnieken	Webtechnologie en textmining
	Proteomics: eiwitstructuren, eiwitfuncties, scheidingstechnieken en data analyse	Transcriptomics: data analyse mbt regulatie van metabole routes	Genomics: annoteren van genomisch DNA	Moleculaire fylogenie: evolutie, multiple sequence alignment mbt signaaltransductie
3e jaar	Stage of minor		Data Mining en grid computing	RNA seq, biostatistiek,
			Eiwitten: carcinogenese en eiwitstructuren	Workflows
4e jaar	Stage, afstudeerstage of minor		Minor of afstudeerstage	

JAAR 1

Basis van programmeren in Python

Complexe structuren

Databases voor opslag van data

	Periode 1	Periode 2	Periode 3	Periode 4
1e jaar	Programmeren in Python (1)	Programmeren in Python (2)	Informatiesystemen met Python	Geautomatiseerde identificatie van eiwitten via sequentievergelijking
	Metabolisme	Opsporen van genetische mutaties bij erfelijke ziektes	Vergelijkende genoomanalyse: evolutie van virussen	Sequentie alignment (BLAST) en functionele eiwit analyse

Ontwikkelen van applicaties voor biologisch analyse en visualisatie op het web

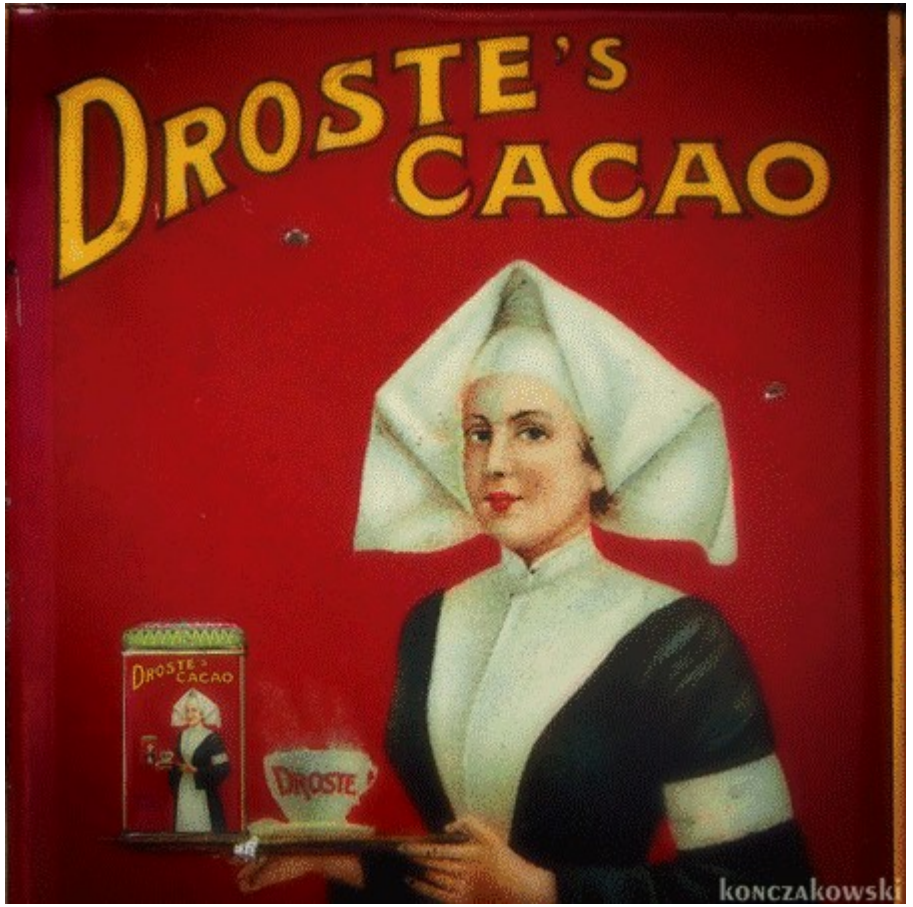
OVERZICHT OWE 1A INFORMATICA

Week	Onderwerpen	
1	IDE: PyCharm Debuggen en testen PEP richtlijnen	PEP tutorial
2	Grafieken met MatPlotLib	MatPlotLib.org
3	Datastructuren	H9 SowP
4	Regular Expressions	H4/H5 DiP
5	Object Oriëntatie	H10 & H11 SowP
6	Recursie	H12 SowP
7	Graphical User Interface	H13 SowP

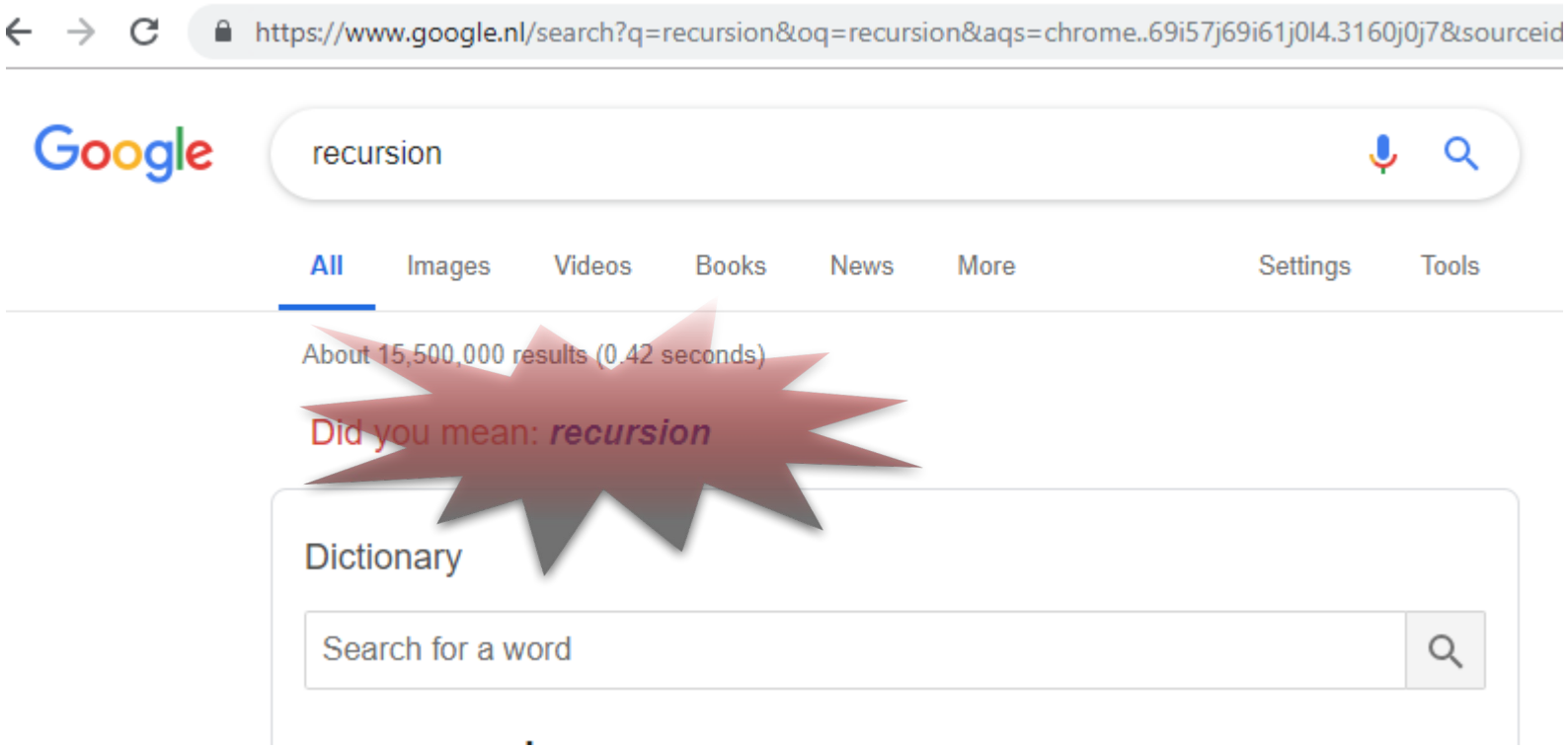
AGENDA

- Introductie recursie
- Voorbeelden recursie
- Recursie vs. iteratie

DROSTE EFFECT



http://en.wikipedia.org/wiki/Recursion 24-maart-2014



google.com 14-december-2018

RECURSIVITEIT

- Recursiviteit is een aanroep aan zichzelf
- Een recursieve functie is een functie die zichzelf aanroept

AGENDA

- Introductie recursie
- Voorbeelden recursie
- Recursie vs. iteratie

BUT FIRST..

Schrijf een stukje code die tekst omdraait.

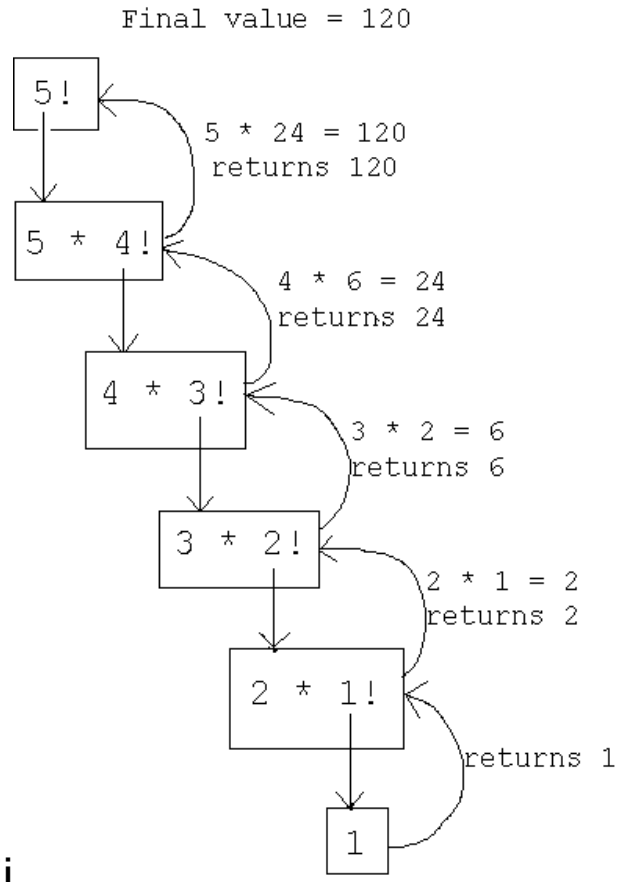
Bijvoorbeeld: Hoi

ioH

REVERSE FUNCTIE RECURSIEF GEDEFINIEERD

```
def backward(text):
    if text == "":
        return text
    else:
        return text[-1] + backward(text[:-1])
```

PROBLEMEN OPLOSSEN MET RECURSIE



Eigen werk auteur

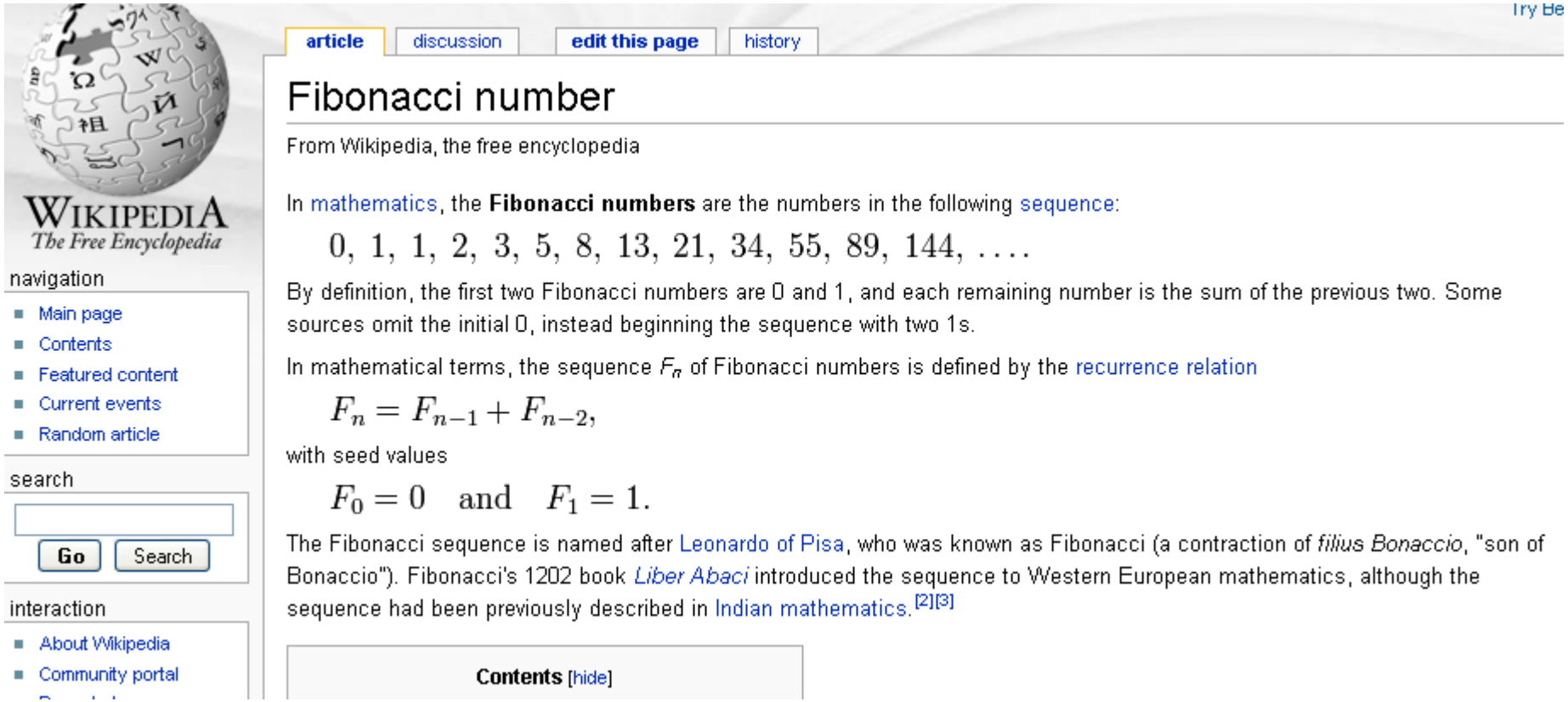
Opdracht

Schrijf een stukje code die de faculteit berekent van (bijv) nummer 5

(dit hoeft nog niet recursief)

```
1   # This program uses recursion to calculate
2   # the factorial of a number.
3
4   def main():
5       # Get a number from the user.
6       number = input('Enter a nonnegative integer: ')
7
8       # Get the factorial of the number.
9       fact = factorial(number)
10
11      # Display the factorial.
12      print 'The factorial of', number, 'is', fact
13
14  # The factorial function uses recursion to
15  # calculate the factorial of its argument,
16  # which is assumed to be nonnegative.
17  def factorial(num):
18      if num == 0:
19          return 1
20      else:
21          return num * factorial(num - 1)
22
23  # Call the main function.
24  main()
```

FIBONACCI NUMBER



article | discussion | edit this page | history

WIKIPEDIA
The Free Encyclopedia

navigation
- Main page
- Contents
- Featured content
- Current events
- Random article

search
Go | Search

interaction
- About Wikipedia
- Community portal

Fibonacci number

From Wikipedia, the free encyclopedia

In mathematics, the **Fibonacci numbers** are the numbers in the following sequence:

$$0,\ 1,\ 1,\ 2,\ 3,\ 5,\ 8,\ 13,\ 21,\ 34,\ 55,\ 89,\ 144,\ \ldots.$$

By definition, the first two Fibonacci numbers are 0 and 1, and each remaining number is the sum of the previous two. Some sources omit the initial 0, instead beginning the sequence with two 1s.

In mathematical terms, the sequence F_n of Fibonacci numbers is defined by the recurrence relation

$$F_n = F_{n-1} + F_{n-2},$$

with seed values

$$F_0 = 0 \quad \text{and} \quad F_1 = 1.$$

The Fibonacci sequence is named after Leonardo of Pisa, who was known as Fibonacci (a contraction of *filius Bonaccio*, "son of Bonaccio"). Fibonacci's 1202 book *Liber Abaci* introduced the sequence to Western European mathematics, although the sequence had been previously described in Indian mathematics.[2][3]

Contents [hide]

wikipedia.com 14-maart-2013

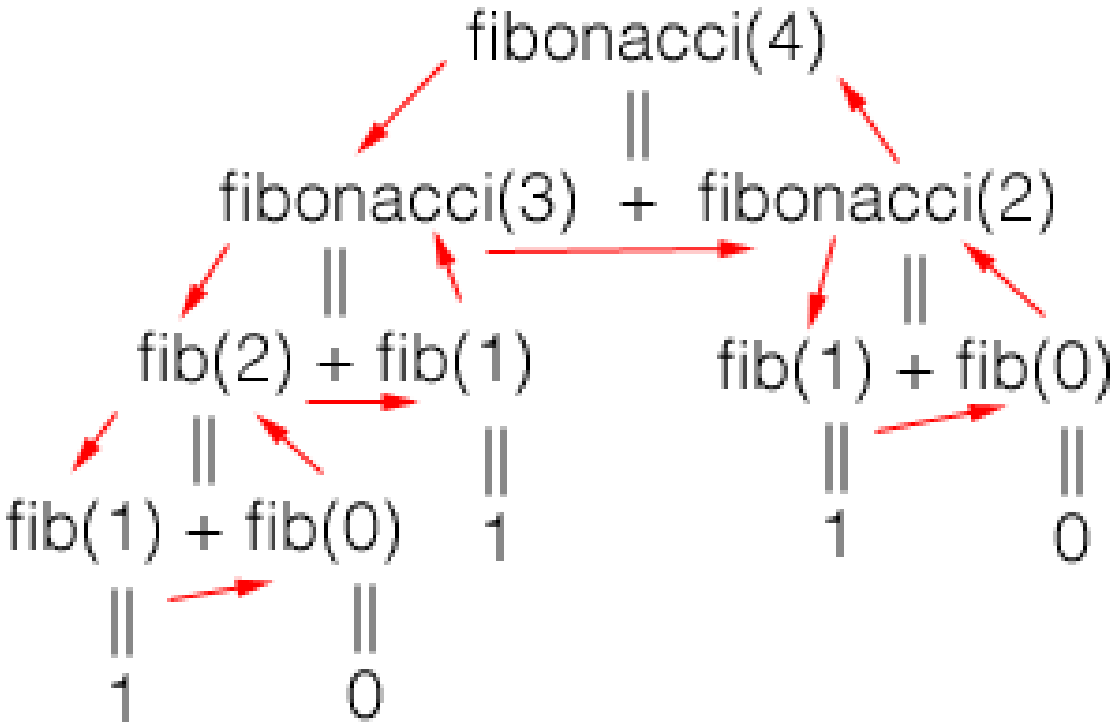
FIBONACCI

Fibonacci

```
# This program uses recursion to print numbers
# from the Fibonacci series.

def main():
    print 'The first 10 numbers in the'
    print 'Fibonacci series are:'

    for number in range(1, 11):
        print fib(number)

# The fib function returns the nth number
# in the Fibonacci series.
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n - 1) + fib(n - 2)

# Call the main function.
main()
```

Eigen werk auteur

FIBONACCI VISUALISATIE RECURSIE



LIMITERINGEN

- Python heeft een beperking op de diepte van recursieve calls
- Er treedt ook vertraging op bij te diepe calls

AGENDA

- Introductie recursie
- Voorbeelden recursie
- Recursie vs. iteratie

RECURSIVITEIT VERSUS ITERATIE

Recursie

1. Minder efficiënt
2. Veel overhead
3. Soms veel duidelijkere oplossing

Looping

1. Efficiënter
2. Minder overhead
3. Soms duidelijkere oplossingen
4. Meestal de beste oplossing

RECURSIE VS. ITERATIE

Recursion

```
>>> def countdown (n):
...     if n <= 0:
...         print 'Blastoff!'
...     else:
...         print n
...         countdown (n - 1)
...
>>> countdown (3)
3
2
1
Blastoff!
>>>
```

For Loop

```
>>> def countdown (n):
...     for i in range (n, -1, -1):
...         if i <= 0:
...             print "Blastoff!"
...         else:
...             print i
...
>>> countdown (3)
3
2
1
Blastoff!
>>>
```

While Loop

```
>>> def countdown (n):
...     while n > 0:
...         print n
...         n = n - 1
...     print "Blastoff!"
...
>>> countdown (3)
3
2
1
Blastoff!
>>>
```

SAMENVATTING

- Recursie is in een aantal situaties een goede oplossing (bijvoorbeeld het doorlopen van de datastructuur tree)
- Meestal zal de iteratieve oplossing sneller en beter zijn

Ieder probleem wat recursief opgelost kan worden kan ook worden opgelost met?

a. Een beslissingsstructuur
b. Een case structuur
c. Een conditionele controle
d. Een iteratieve controle

Wat zal onderstaande code weergeven?

a. None
b. ATGC
c. A
d. C

```
def main():
  print show_me("ATGC")

def show_me(arg):
  if len(arg)>1:
    return show_me (arg[1:])
  else:
    return arg
main()
```

Wat retourneert onderstaande functie?

a) 4
b) 24
c) 36
d) 12

```
def x(n):
  if n == 0 or n == 1:
    return 1
  else:
    return n * x(n - 1)

print x(4)
```

Een recursieve oplossing zal vergeleken met een loop meestal:

a) Veel trager zijn
b) Veel sneller zijn
c) Even snel zijn

VERANTWOORDING

- In deze uitgave is géén auteursrechtelijk beschermd werk opgenomen
- Alle teksten © Teuntje Peeters/HAN tenzij expliciet externe bronnen zijn aangegeven
- Screenshots op basis van eigen werk auteur en/of vernoemde sites
- Eventuele images zijn opgenomen met vermelding van bron