# Programmeermethoden

Algoritmen, talen, ...

Walter Kosters en Jonathan Vis

week 13: 4–8 december 2023

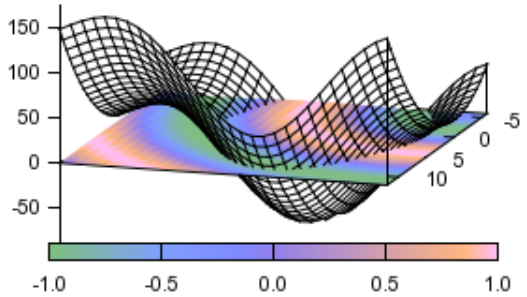
www.liacs.leidenuniv.nl/~kosterswa/pm/

Gomoku programmeren we als volgt:

- week 1 (“10”): pointerpracticum, opgave lezen
- week 2 (“11”): klassen, pointerbord, meerdere files, ruw spelen
- week 3 (“12”): spel helemaal in orde maken, stapel
- week 4 (“13”): (vervolgpartijen), experiment (`gnuplot`), verslag

`www.liacs.leidenuniv.nl/~kosterswa/pm/op4pm.php`

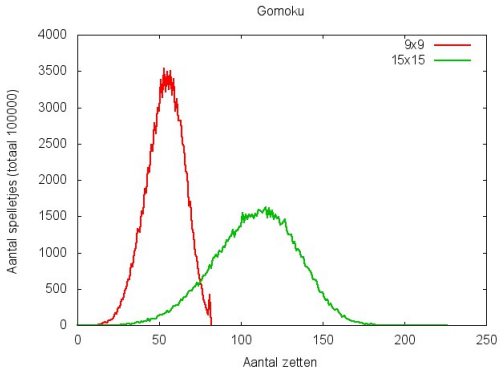
Gebruik `gnuplot` om een eenvoudige grafiek te maken, ook in Windows:




```
gnuplot> plot "stats.txt" with lines
```

Hierbij is de file `stats.txt` zoiets als:

```
1   7
2   12
3   14
```




Zie www.gnuplot.info.

Op allerlei colleges en in allerlei boeken en artikelen worden algoritmen behandeld, bijvoorbeeld bij de volgende Informatica-colleges in Leiden (semester tussen haakjes):

- Programmeermethoden (1), Algoritmiek (2)
- Datastructuren (3), Kunstmatige intelligentie (4)
- . . .

Je kunt algoritmen op allerlei manieren rubriceren, bijvoorbeeld met behulp van de volgende begrippen:

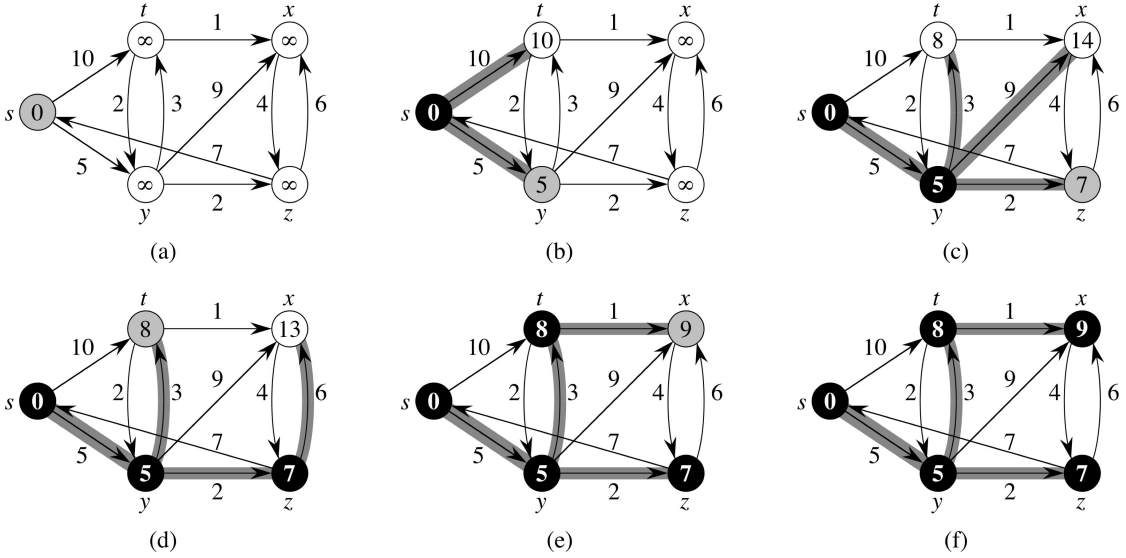
verdeel en heers, numerieke wiskunde, graafalgoritmen, dynamisch programmeren, patroonherkenning, adversary, data mining, geometrisch modelleren, backtracking, benaderende algoritmen, kunstmatige intelligentie, neurale netwerken, evolutionaire algoritmen, $\mathcal{P} \leftrightarrow \mathcal{NP}$, gretige algoritmen, snelle Fouriertransformatie,

. . .

We bekijken er een paar.

Gegeven een graaf $G$, met afstanden op de takken, en twee knopen $a$ en $b$. Gevraagd: kortste pad van $a$ naar $b$.

Oplossing: het algoritme van Dijkstra.



Als $n$ een priemgetal is, geldt dat $a^{n-1} - 1$ deelbaar is door $n$ voor $a = 1, 2, \ldots, n-1$. Het omgekeerde is bijna waar.

Dit suggereert het volgende algoritme dat "bepaalt" of $n$ een priemgetal is: Als $2^{n-1} - 1$ niet deelbaar is door $n$ is $n$ zeker geen priemgetal, en anders (misschien) wel. Dit gaat fout bij 341, 561, ..., maar dat is te verbeteren: probeer andere $a$; echter, 561, een Carmichael-getal, blijft lastig. (Uiteindelijk: Miller-Rabin.)

Het algoritme is een randomized algoritme, en wel een Monte Carlo algoritme: het ene antwoord is altijd juist, het anders soms niet. Bij Las Vegas algoritmen zijn de antwoorden altijd juist — maar het duurt soms lang.

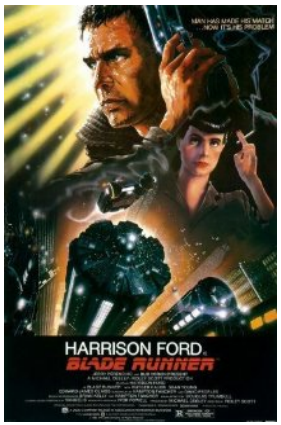
En hoe maak je een willekeurige permutatie, dat wil zeggen, een random volgorde van de getallen $1, 2, \ldots, n$?

```
// stop random permutatie van 0,1,...,n-1 in array A
void maakpermutatie (int A[ ], int n) {
  int i;  // array-index
  int r;  // random array-index
  for ( i = 0; i < n; i++ ) A[i] = i;
  for ( i = n-1; i >= 0; i-- ) {
    r = rand ( ) % ( i+1 );  // 0 <= r <= i, random
    wissel (A[i],A[r]);
  }//for
}//maakpermutatie
```

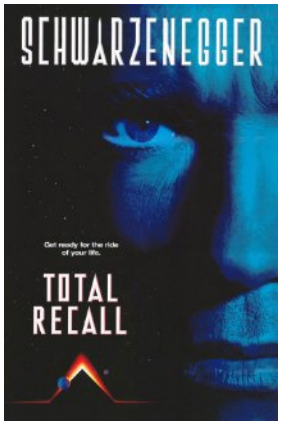
`rand ( )` geeft een random-getal; `srand (42)` zet het "seed".

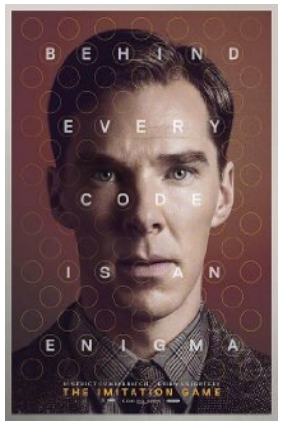
# Kunstmatige intelligentie $\stackrel{?}{=}$ algoritmen

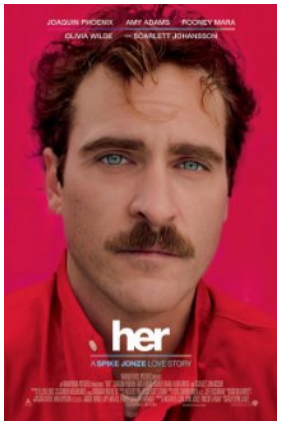


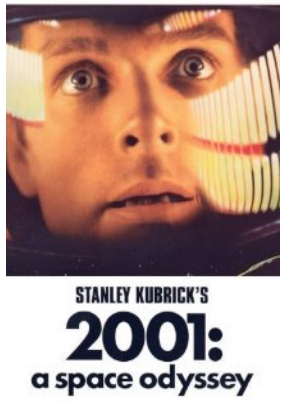




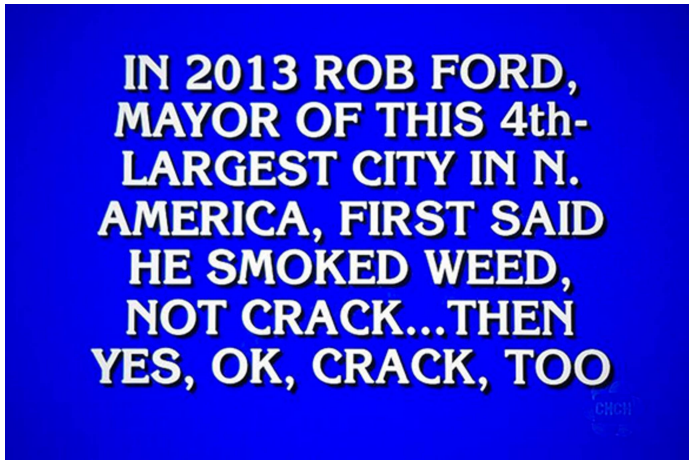


# IBM: Jeopardy!



2011


# DeepNash

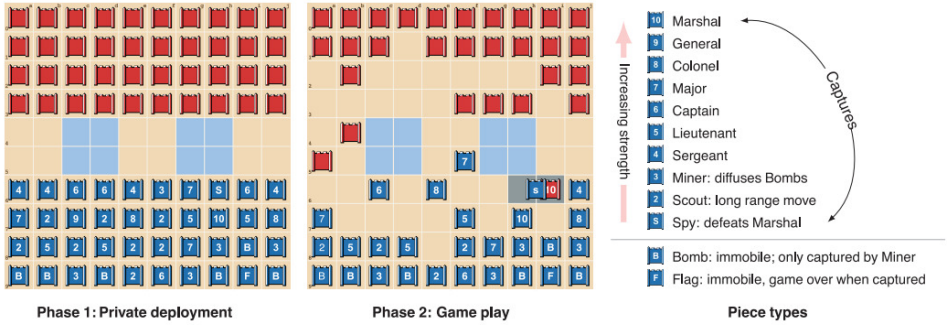
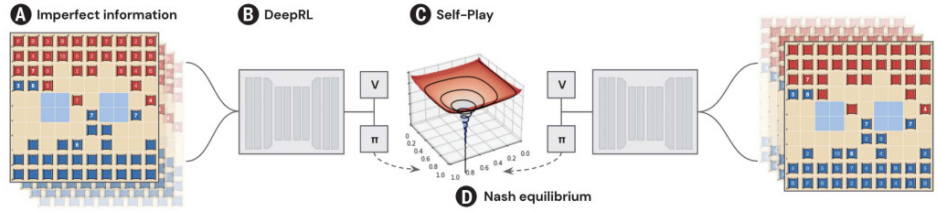



**Fig. 1. Stratego is a two-player board game in which players try to capture the opponent's flag.** Initially, the players secretly deploy 40 pieces of diverse strengths on the board. Then, they take turns moving pieces, possibly encountering an opponent piece that reveals both piece identities, and then the weaker piece is removed. Two lakes (indicated in blue) cannot be crossed by any piece. The complete rules are defined by the International Stratego Federation.




Replicator dynamics: $\frac{d}{d\tau}\pi^i_\tau(a^i) = \pi^i_\tau(a^i)\left[Q^i_{\pi_\tau}(a^i) - \sum_{b^i}\pi^i_\tau(b^i)Q^i_{\pi_\tau}(b^i)\right]$

Reward transformation: $r^i(\pi^i, \pi^{-i}, a^i, a^{-i}) = r^i(a^i, a^{-i}) - \eta\log\left(\frac{\pi^i(a^i)}{\pi^i_{\text{reg}}(a^i)}\right) + \eta\log\left(\frac{\pi^{-i}(a^{-i})}{\pi^{-i}_{\text{reg}}(a^{-i})}\right)$

breaking news

↓

RESEARCH

MACHINE LEARNING

**Mastering the game of Stratego with model-free multiagent reinforcement learning**

Julien Perolat*†, Bart De Vylder*†, Daniel Hennes, Eugene Tarassov, Florian Strub, Vincent de Boer‡, Paul Muller, Jerome T. Connor, Neil Burch, Thomas Anthony, Stephen McAleer, Romuald Elie, Sarah H. Cen, Zhe Wang, Audrunas Gruslys, Aleksandra Malysheva, Mina Khan, Sherjil Ozair, Finbarr Timbers, Toby Pohlen, Tom Eccles, Mark Rowland, Marc Lanctot, Jean-Baptiste Lespiau, Bilal Piot, Shayegan Omidshafiei, Edward Lockhart, Laurent Sifre, Nathalie Beauguerlange, Remi Munos, David Silver, Satinder Singh, Demis Hassabis, Karl Tuyls*†

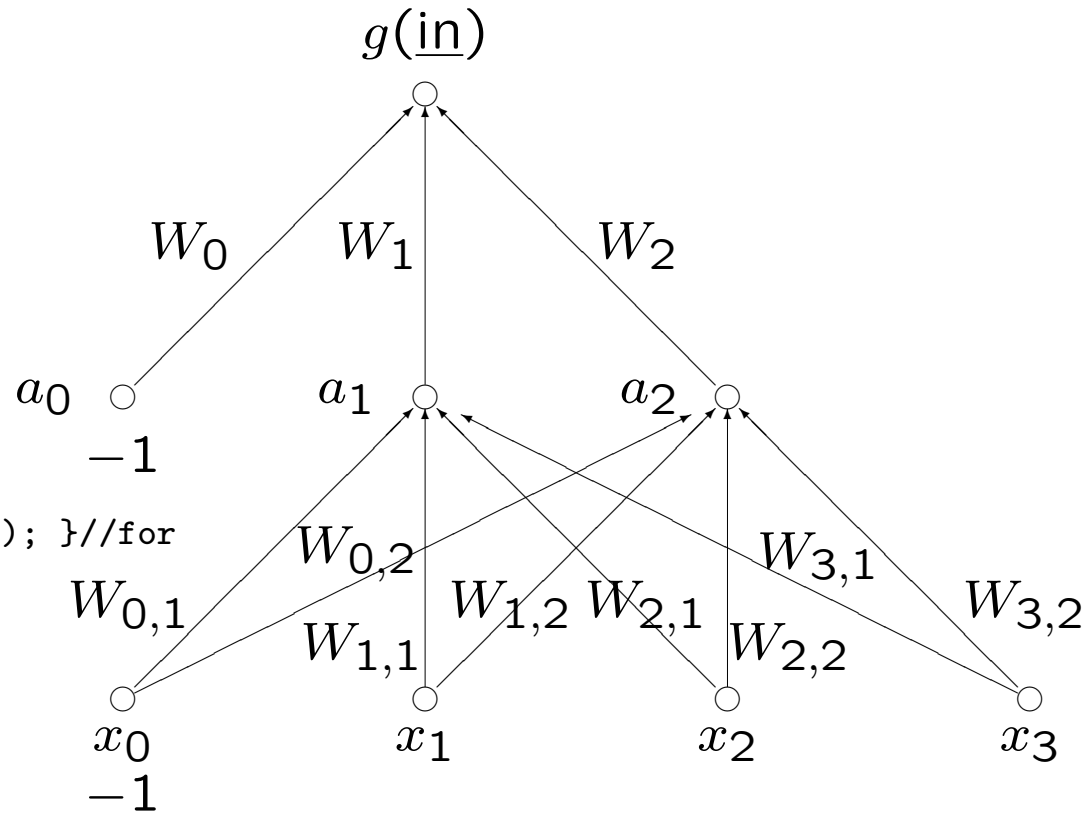
We introduce DeepNash, an autonomous agent that plays the imperfect information game Stratego at a human expert level. Stratego is one of the few iconic board games that artificial intelligence (AI) has not yet mastered. It is a game characterized by a twin challenge: It requires long-term strategic thinking as in chess, but it also requires dealing with imperfect information as in poker. The technique underpinning DeepNash uses a game-theoretic, model-free deep reinforcement learning method, without search, that learns to master Stratego through self-play from scratch. DeepNash beat existing state-of-the-art AI methods in Stratego and achieved a year-to-date (2022) and all-time top-three ranking on the Gravon games platform, competing with human expert players.


Perolat et al., Science 378 (2022) 990–996: Mastering the game of Stratego . . .

Deep neural nets, reinforcement learning, selfplay.

Kunnen computers denken? Diplomacy! ChatGPT! link

```
for ( j = 1; j <= hs; j++ ) {
  s[j] = -ItH[0][j];
  for ( k = 1; k <= ip; k++ )
    s[j] += ItH[k][j] * I[k];
  HtA[j] = g (s[j]); }//for
for ( i = 0; i < os; i++ ) {
  sO = -HtO[0][i];
  for ( j = 1; j <= hs; j++ )
    sO += HtO[j][i] * HtA[j];
  O[i] = g (sO);
  dO[i] = gp (sO) * ( T[i] - O[i] ); }//for
for ( j = 1; j <= hs; j++ ) {
  d[j] = 0;
  for ( i = 0; i < os; i++ )
    d[j] += HtO[j][i] * dO[i];
  d[j] *= gp (sum[j]); }//for
for ( j = 0; j <= hs; j++ )
  for ( i = 0; i < os; i++ )
    HtO[j][i] += a * HtA[j] * dO[i];
for ( k = 0; k <= ip; k++ )
  for ( j = 1; j <= hs; j++ )
    ItH[k][j] += a * I[k] * d[j];
```



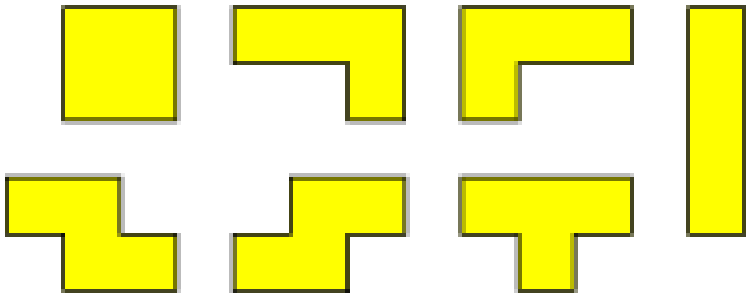

matrix-vermenigvuldiging, GPU's

Aan een spel als Tetris kleven allerlei vragen:

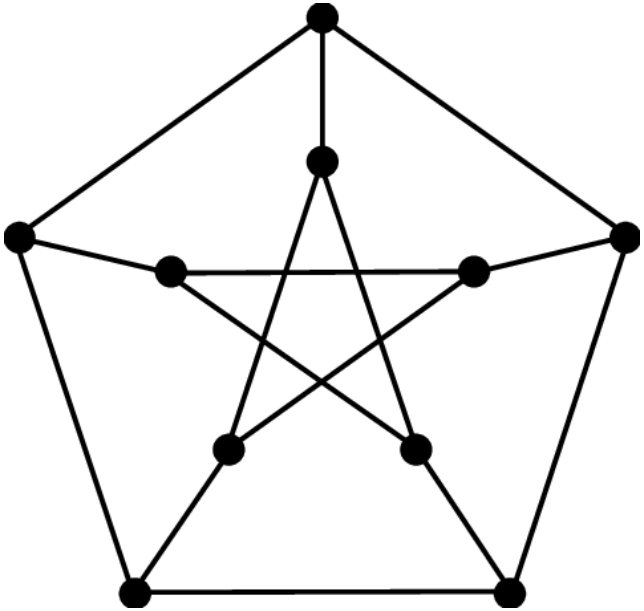
- Hoe speel je het zo goed mogelijk?
  (AI = Kunstmatige intelligentie)
- Hoe moeilijk is het? (complexiteit)
- Wat kan er allemaal gebeuren?

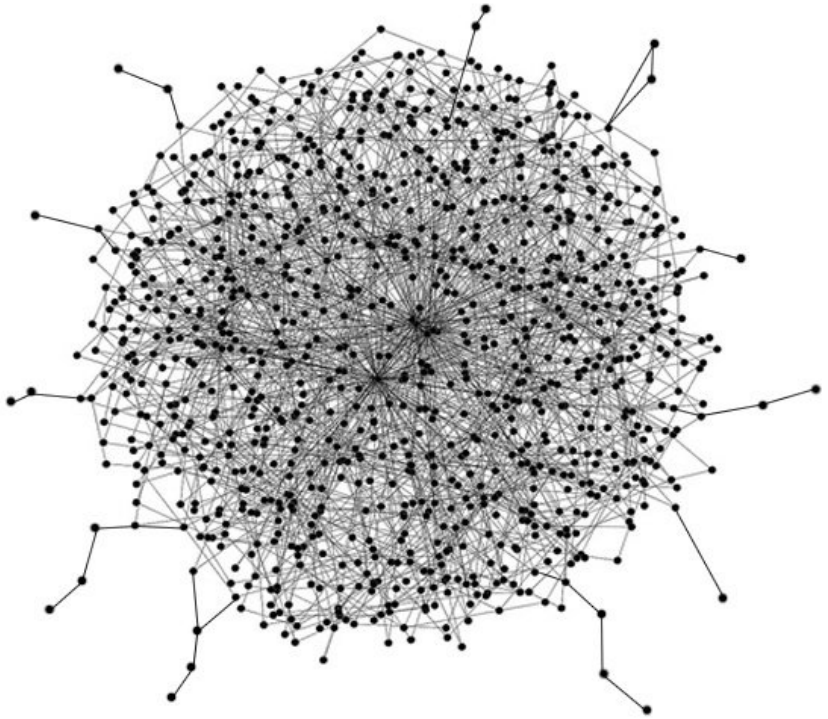
Zo is bijvoorbeeld bewezen dat sommige Tetris-vragen $\mathcal{NP}$-volledig zijn (gezamenlijk werk met mensen van MIT), dat je bijna alle configuraties kunt bereiken, maar dat niet alle problemen "beslisbaar" zijn.

De 7 Tetris-stukken:



De vraag "Kun je met een gegeven serie (inclusief volgorde) van deze stukken een deels al gevuld bord helemaal leeg spelen?" is $\mathcal{NP}$-volledig.
Als iemand het bord leeg speelt kun je dat eenvoudig controleren. Als het *niet* kan, kan men (tot nu toe) niks beters verzinnen dan alle mogelijkheden één voor één na te gaan. En dat zijn er veel!





Is de graaf samenhangend? ($\mathcal{P}$)

Heeft de graaf een Hamilton-circuit? ($\mathcal{NP}$)

$123456789 \times 987654321 = 121932631112635269$? ($\mathcal{P}$)

$\mathcal{P}$ en $\mathcal{NP}$ zijn "klassen" van beslissingsproblemen oftewel ja-nee-problemen. Het probleem of een graaf samenhangend is, zit in $\mathcal{P}$. Idem voor het probleem of de som van getallen $x$ en $y$ gelijk is aan getal $z$. Het probleem of een graaf een Hamilton-circuit heeft, zit in $\mathcal{NP}$, en is zelfs "$\mathcal{NP}$-volledig"; je kunt het "eenvoudig", maar niet efficiënt, oplossen met bruteforce technieken. En een "ja" is met een juiste oplossing eenvoudig te controleren.

$\mathcal{P}$ is de klasse van beslissingsproblemen die door een "deterministische Turing-machine" in "polynomiale tijd" (in de grootte van de invoer) kunnen worden opgelost.
$\mathcal{NP}$ is de klasse van beslissingsproblemen die door een "niet-deterministische Turing-machine" in "polynomiale tijd" kunnen worden opgelost: je mag "gokken".

Open probleem: geldt $\mathcal{P} = \mathcal{NP}$?

Naast C++ (C: Brian Kernighan, Dennis Ritchie, Ken Thompson, ± 1970; C++: Bjarne Stroustrup, ± 1985, nu C++20), wordt ook veel MATLAB gebruikt, en R, en Java. En Qt voor interfaces. En . . . Python.

Ritchie en Thompson in 1972

De programmeertaal Java lijkt veel op C++.
Enkele belangrijke verschillen:

- Java is (nog) meer object-georienteerd
- Java heeft automatische garbage collection
- Java heeft (bijna) geen pointers
- Java is platform-onafhankelijk
- Java kent applets: WWW-applicaties, met GUI's
- Java-code wordt *gecompileerd* naar bytecode, en daarna *geinterpreteerd* door de Java Virtual Machine (JVM)

Python, ook zeer geschikt voor scripting en "prototyping", is ontwikkeld door Guido van Rossum uit Nederland. Er zit veel overlap in met de andere genoemde talen.

Voor meer informatie (zelfstudie), zie:

www.liacs.leidenuniv.nl/~kosterswa/pm/pythonextra.php

- interpreteren, niet compileren: `python hello.py`

```
def hello():
    print("Hello world!")

hello()
```

- interactieve mode
- type van variabele kan eenvoudig wijzigen
- indentatie
- NumPy, Matplotlib, . . .

```
# Dit is een regel met commentaar
import math # voor "pi"
print("Geef straal, en Enter .. ",end="")
straal = float(input())
if straal > 0:
    print("Oppervlakte: ",end="")
    print(math.pi * straal * straal)
else:
    print("Niet zo negatief ...")
print("Einde van dit programma.")
exit(0)
```

Geen accolades, geen punt-komma's. Dit is overigens Python3; er zijn subtiele verschillen met oudere versies.

Interactieve mode, via `python3`:

```
>>> a, b, c = 1, 7.1234, "een string" # gebruik " of '
>>> print(type(a),type(b),type(c))
<class 'int'> <class 'float'> <class 'str'>
>>> a = "wat anders"
>>> print(type(a),a)
<type 'str'> 'wat anders'
>>> 42**97  # machtsverheffen
285220783529230204153175149217567486191944890823858197365940414630662077361378060803659240001284703557970677097414100792138630484270254055597330734645457715 2L
>>> a+d  # gaat fout:
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'd' is not defined
>>> Ctrl-D
```

Let op de indentatie, oftewel het inspringen:

```
if y >= 3 and ( x == 4 or x == 5 ): # and voor && ...
    pass # doe niets
elif z == 12:  # in plaats van else if uit C++
    x = 0
    y = 678
else:
    print("Hoe verzin je het!")


for karakter in ['a', 'e', 'i', 'o', 'u']:
    print(karakter,end="")
som = 0
for i in range(6):  # i = 0,1,2,3,4,5
    som = som + i*i
```

# Functies & Lezen uit file

```
def telop(a,b):
    c = a + b
    return c

q = telop(12345,6789)


f = open("mijnfile.txt","r")
for regel in f:
    print(regel,end="")
f.close()
```

Parameter-overdracht in Python is call-by-object-reference:

```
def vergroot(lijst):
    lijst += [10,20,30,40]

def vernieuw(lijst):
    lijst = [1000,1001]

lijst = [7,8]   # een "mutable" object
vergroot(lijst)
# nu is lijst [7,8,10,20,30,40]
vernieuw(lijst)
# en nu is lijst nog steeds [7,8,10,20,30,40]
```

En integers zijn “immutable”.

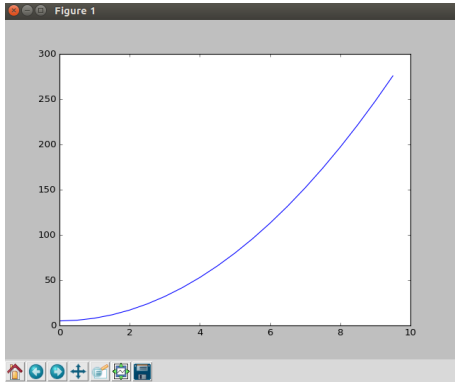
```
def simpelsort(A):
    for i in range(len(A)):
    # Zoek kleinste element in ongesorteerde stuk [i:]
        kl = i
        for j, el in enumerate(A[i:]):
            if el < A[kl]:
                kl = i + j
        # Wissel om
        if i != kl:
            A[i], A[kl] = A[kl], A[i]

# Test
B = [47, 10, 7, 3, 31, 75, 18, 21, 48, 79]
simpelsort(B)
print(B)
```

```
import numpy as np  # hier zitten ook arrays in!
import matplotlib.pyplot as plt

# Bepaal de x-coordinaten waarvoor we willen plotten
x = np.arange(0, 10, 0.5)
# Bereken nu voor elk x-coordinaat de y-waarde
# Functie: y = 3x^2 + 5
y = 3 * x * x + 5

# Geef de x- en y-arrays als parameters aan de plot-functie
plt.plot(x, y)
# Zet de plot op het scherm
plt.show()
exit(0)
```



- werk aan de vierde programmeeropgave — de deadline is op maandag 11 december 2023 (vragenuur!)

- volgende week laatste college en werkcollege: donderdag 14 december 2023: Oude tentamens

- tentamen:
  woensdag 17 januari 2024, 13:00–16:00 uur,
  Universitair Sportcentrum; aanmelden!
  hertentamen:
  donderdag 28 maart 2024, 13:00–16:00 uur

- www.liacs.leidenuniv.nl/~kosterswa/pm/