# Programmeermethoden NA

# Week 9: NumPy & Matplotlib

Kristian Rietveld

http://liacs.leidenuniv.nl/~rietveldkfd/courses/prna2016/

Universiteit Leiden
The Netherlands

# Arrays met meerdere dimensies

- Vorige week hebben we kennis gemaakt met NumPy en het werken met 2-dimensionale arrays.

```
>>> M = np.tile(6, (3, 4))    # 3 rijen, 4 kolommen
>>> print M
[[6 6 6 6]
 [6 6 6 6]
 [6 6 6 6]]
```

# Indexeren & Slicen

Gegeven:

```
>>> A
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24]])
```

Voorbeelden:

```
A[3,4]           = 19

A[3,:]           = array([15, 16, 17, 18, 19])

A[3:5,1:3]       = array([[16, 17],
                          [21, 22]])
```

# Diagonaal aanpassen

```
>>> A = np.eye(6)
>>> np.diag(A)
array([ 1.,  1.,  1.,  1.,  1.,  1.])
# Maakt een kopie! Aanpassen helpt niet.
>>> d = np.diag(A)
# Geeft een fout:
>>> np.diag(A) = np.arange(10, 16)
```

Hoe pakken we dat dan aan?

# Diagonaal aanpassen (2)

```
>>> A[range(6),range(6)] = range(10, 16)
>>> A
array([[ 10.,   0.,   0.,   0.,   0.,   0.],
       [  0.,  11.,   0.,   0.,   0.,   0.],
       [  0.,   0.,  12.,   0.,   0.,   0.],
       [  0.,   0.,   0.,  13.,   0.,   0.],
       [  0.,   0.,   0.,   0.,  14.,   0.],
       [  0.,   0.,   0.,   0.,   0.,  15.]])
```

# Multi-dimensionale arrays maken

Het is ook mogelijk een multi-dimensionale array te maken vanuit een geneste lijst.

```
>>> C = np.array([[1, 2, 3], [6, 7, 4]])
>>> print C
[[1 2 3]
 [6 7 4]]
>>> print C.shape
(2, 3)
```

Of aan de hand van een string in "Matlab-notatie":

```
>>> D = np.array(np.mat("1 2 3; 6 7 1"))
>>> print D
[[1 2 3]
 [6 7 1]]
```

# Het veranderen van de vorm

- Met `.reshape()` kun je de vorm van een array aanpassen. De parameter is een vorm-tuple met de gewenste vorm.
- Let op: het aantal elementen blijft hetzelfde.

```
>>> print np.arange(10, 20).reshape((2, 5))
[[10 11 12 13 14]
 [15 16 17 18 19]]
>>> print np.arange(10, 20).reshape((5, 2))
[[10 11]
 [12 13]
 [14 15]
 [16 17]
 [18 19]]
```

# Werken met verschillende vormen arrays

- Operaties kunnen alleen worden uitgevoerd op arrays met dezelfde vorm.
- Met `.reshape()` kunnen we een array een andere vorm geven om een operatie toch mogelijk te maken.

```
>>> A = np.ones( (4, 3) )
>>> B = np.array( [1, 2, 3] )  # shape: (3, )
# B optellen bij elke rij van A.
>>> A + B.reshape( (1, 3) )
array([[ 2.,  3.,  4.],
       [ 2.,  3.,  4.],
       [ 2.,  3.,  4.],
       [ 2.,  3.,  4.]])
```

# Blokhaken

Let op de extra blokhaak corresponderend met de extra dimensie:

- Shape (3,): `[1, 2, 3]`
- Shape (1, 3): `[[1, 2, 3]]`

In het voorbeeld probeert NumPy standaard een dimensie toe te voegen: (3,) wordt omgezet naar (1, 3) om een operatie toch mogelijk te maken.

- `A + B.reshape( (1,3) )` en `A + B` in dit geval hetzelfde.

# Gebruik kolomvectoren

- Een kolomvector is eigenlijk 2-dimensionaal: (4, 1).
- Als we een 1-dimensionale array als kolomvector willen gebruiken, moeten we dus reshape toepassen.
- Ook mag de volgende notatie om een dimensie toe te voegen (`np.newaxis`).

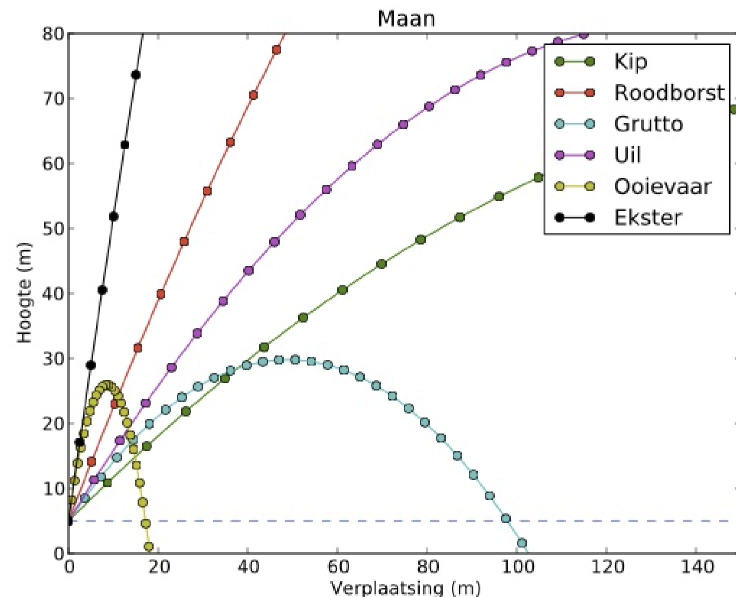
```
>>> C = np.array( [1, 2, 3, 4] )
>>> A + C.reshape( (4, 1) )
array([[ 2.,  2.,  2.],
       [ 3.,  3.,  3.],
       [ 4.,  4.,  4.],
       [ 5.,  5.,  5.]])
>>> A + C[:,np.newaxis]
array([[ 2.,  2.,  2.],
       [ 3.,  3.,  3.],
       [ 4.,  4.,  4.],
       [ 5.,  5.,  5.]])
```

# Nu eerst

Zo direct meer NumPy, we kijken nu eerst naar het maken van plots.

# matplotlib

- Matplotlib is een plotting "package" waarmee hoge kwaliteit plots kunnen worden gemaakt.
- Zeer veel mogelijkheden.
- Wordt gebruikt in combinatie met NumPy.



# Een eerste plot

```
import numpy as np
import matplotlib.pyplot as plt

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

# Kleuren en markers

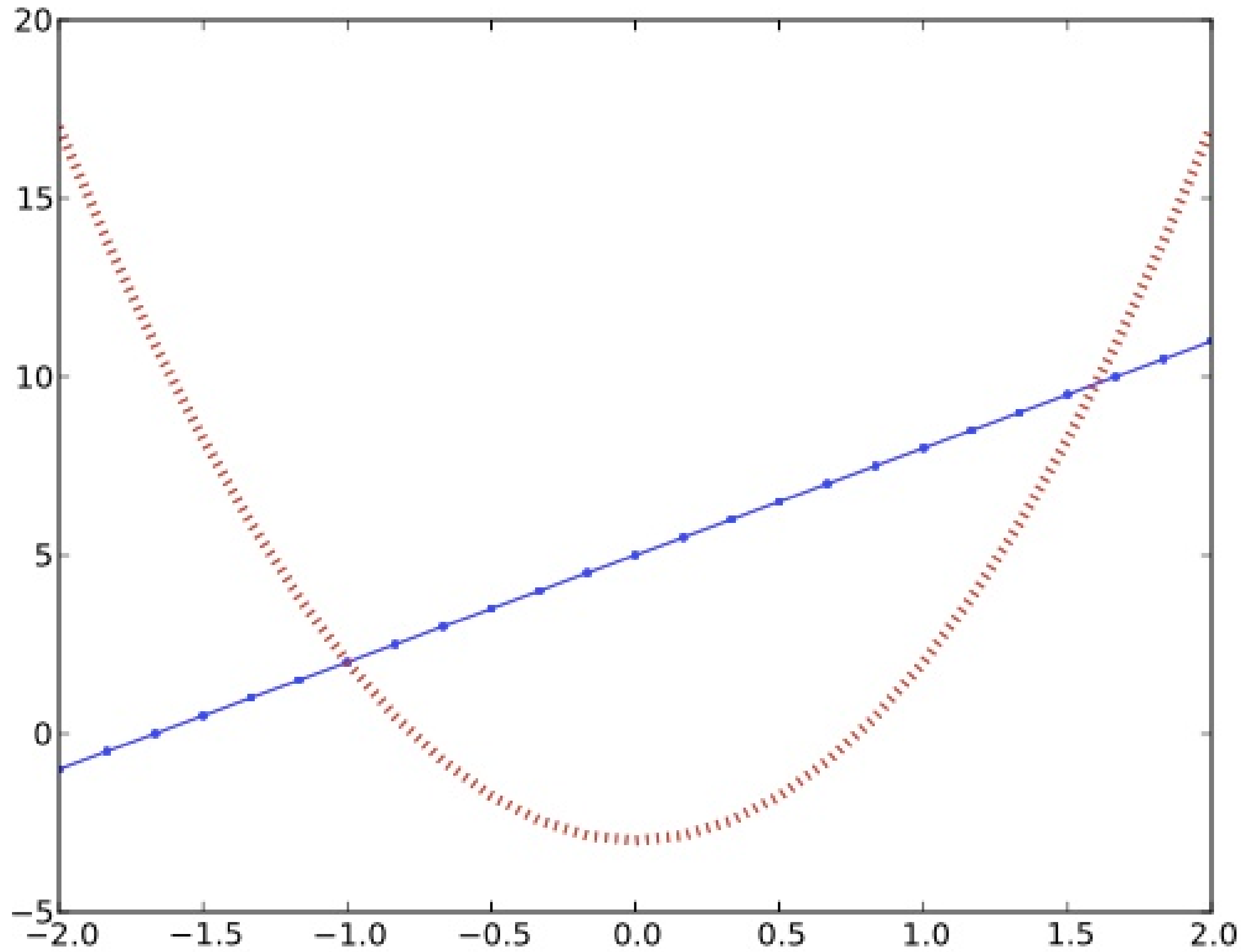
`plt.plot( )` accepteert een groot aantal argumenten:

- `color="red"`
- `marker="o"` - punten markeren met cirkels.
- `linewidth=2.5` - dikke lijn.
- `linestyle="dotted"` - stippellijn.
- `label="Mijn lijn"` - komt in de legenda terecht.

# Kleuren en markers (2)

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-2, 2, 25)
y1 = 3 * x + 5
y2 = 5 * x ** 2 - 3

plt.plot(x, y1, color="blue", lw=1.0,
         linestyle="solid", marker=".")
plt.plot(x, y2, color="red", lw=4.0,
         linestyle="dotted")

plt.show()
```

# Kleuren en markers (3)



# Titel & labels

- Zonder titel en aslabels is de plot natuurlijk niet af.
- `plt.title("titel")`: titel van de plot.
- `plt.xlabel("label")`, `plt.ylabel("label")`: aslabels.
- We mogen TeX gebruiken in matplotlib strings

# Grid en assen

- Met `plt.grid(True)` kun je een achtergrond grid aanzetten.
- De intervallen van de assen kunnen op verschillende manieren worden ingesteld:
  - `plt.ylim(-2, 10)` en analoog voor `plt.xlim()`.
  - Of: `plt.axis(xmin=0, xmax=20., ymin=-10, ymax=100.)`.
- `plt.xscale("log")`: geef de x-as een logaritmische schaal.

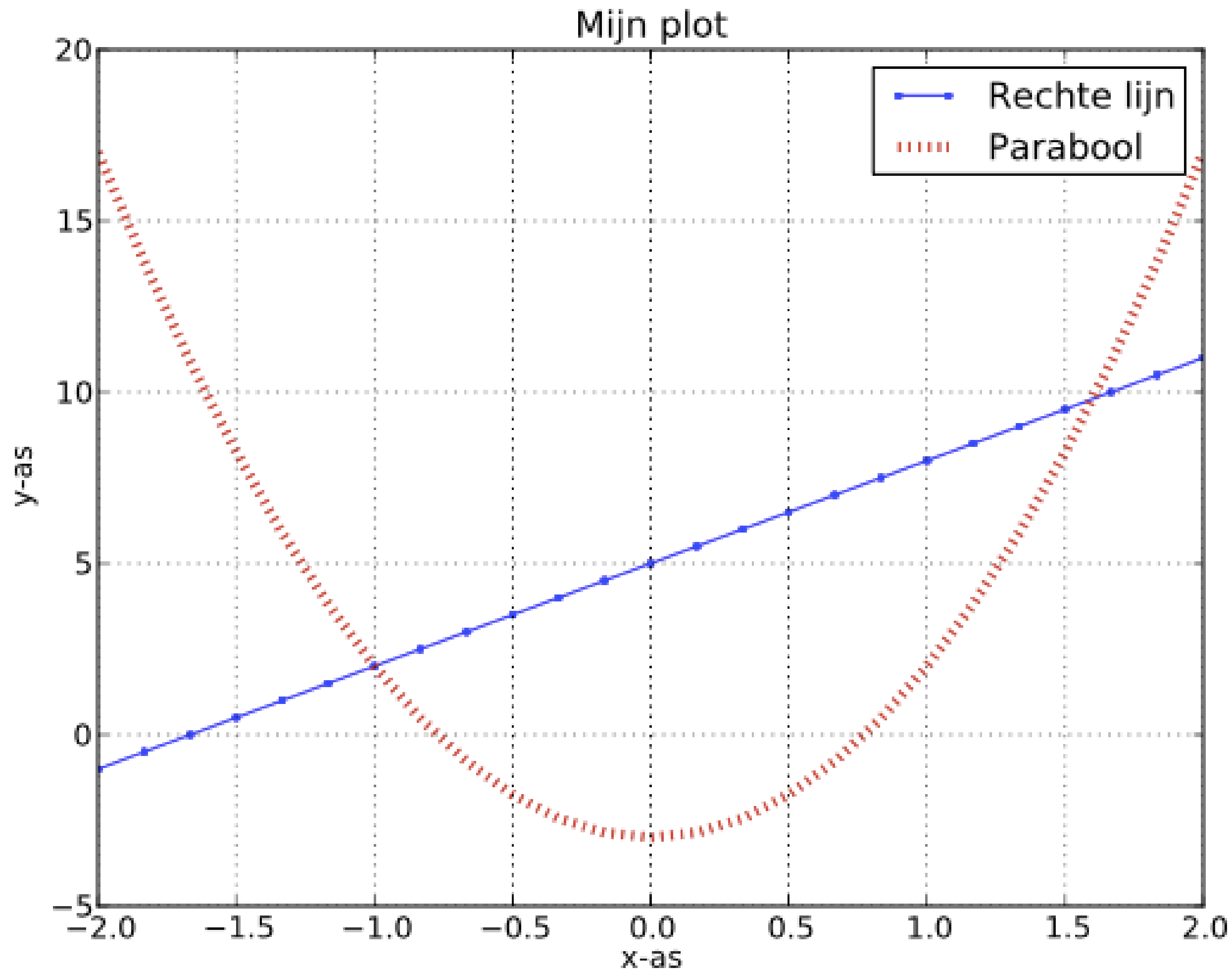
# Legenda

- De opgegeven labels kunnen eenvoudig in een legenda worden afgebeeld.
- `plt.legend(loc="upper right")`.
- Je mag ook opgeven iets als `center`, `lower left`, etc.

# Voorbeeld

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-2, 2, 25)
y1 = 3 * x + 5
y2 = 5 * x ** 2 - 3

plt.plot(x, y1, color="blue", lw=1.0,
         linestyle="solid", marker=".",
         label="Rechte lijn")
plt.plot(x, y2, color="red", lw=4.0,
         linestyle="dotted",
         label="Parabool")

plt.title("Mijn plot")
plt.xlabel("x-as")
plt.ylabel("y-as")

plt.grid(True)

plt.legend(loc="upper right")

plt.show()
```

# Voorbeeld



# Opslaan naar een bestand

- Om op te slaan als PDF bestand: vervang `plt.show()` met `plt.savefig("hallo.pdf")`.

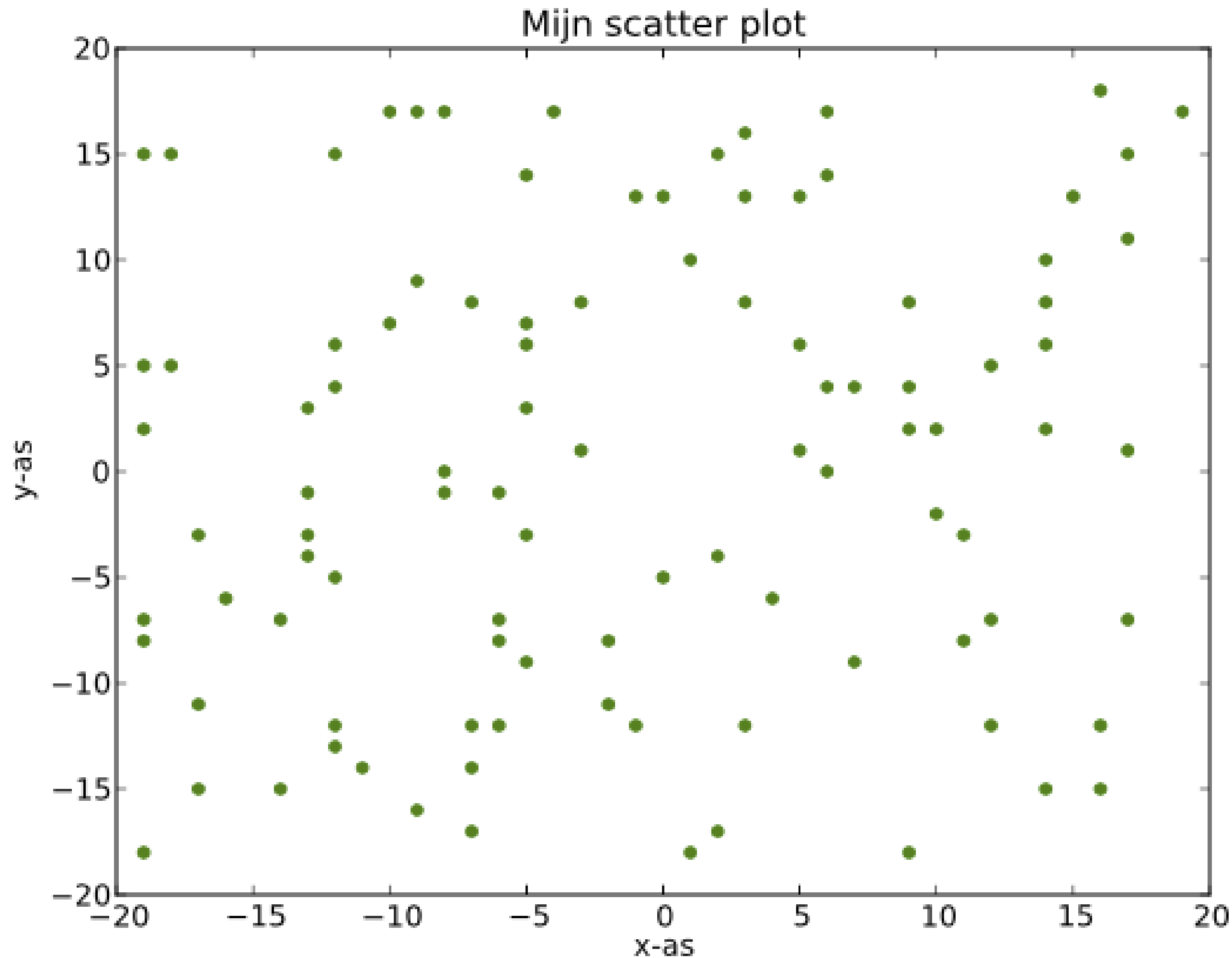
# Meerdere plots maken

- Herhaalde aanroepen van `plt.plot()` tekenen in hetzelfde figuur.
- Met de functie `plt.figure()` kunnen we een nieuw figuur beginnen.
- Voor het maken van meerdere plots volg je de volgende stappen:
  - `plt.figure()`.
  - Een of meerdere aanroepen `plt.plot()`.
  - Plot opmaken door assen in te stellen, titel te zetten, enz.
  - `plt.show()` of `plt.savefig()`.
  - Optioneel: terug naar stap 1 voor de volgende plot.

# Scatter plots

```
import numpy as np
import matplotlib.pyplot as plt

x = np.random.random_integers(-19, 19, 100)
y = np.random.random_integers(-19, 19, 100)

plt.scatter(x, y, color="green")

plt.title("Mijn scatter plot")
plt.xlabel("x-as")
plt.xlim(-20, 20)

plt.ylabel("y-as")
plt.ylim(-20, 20)

plt.savefig("scatterplot.pdf")
```

# Scatter plots

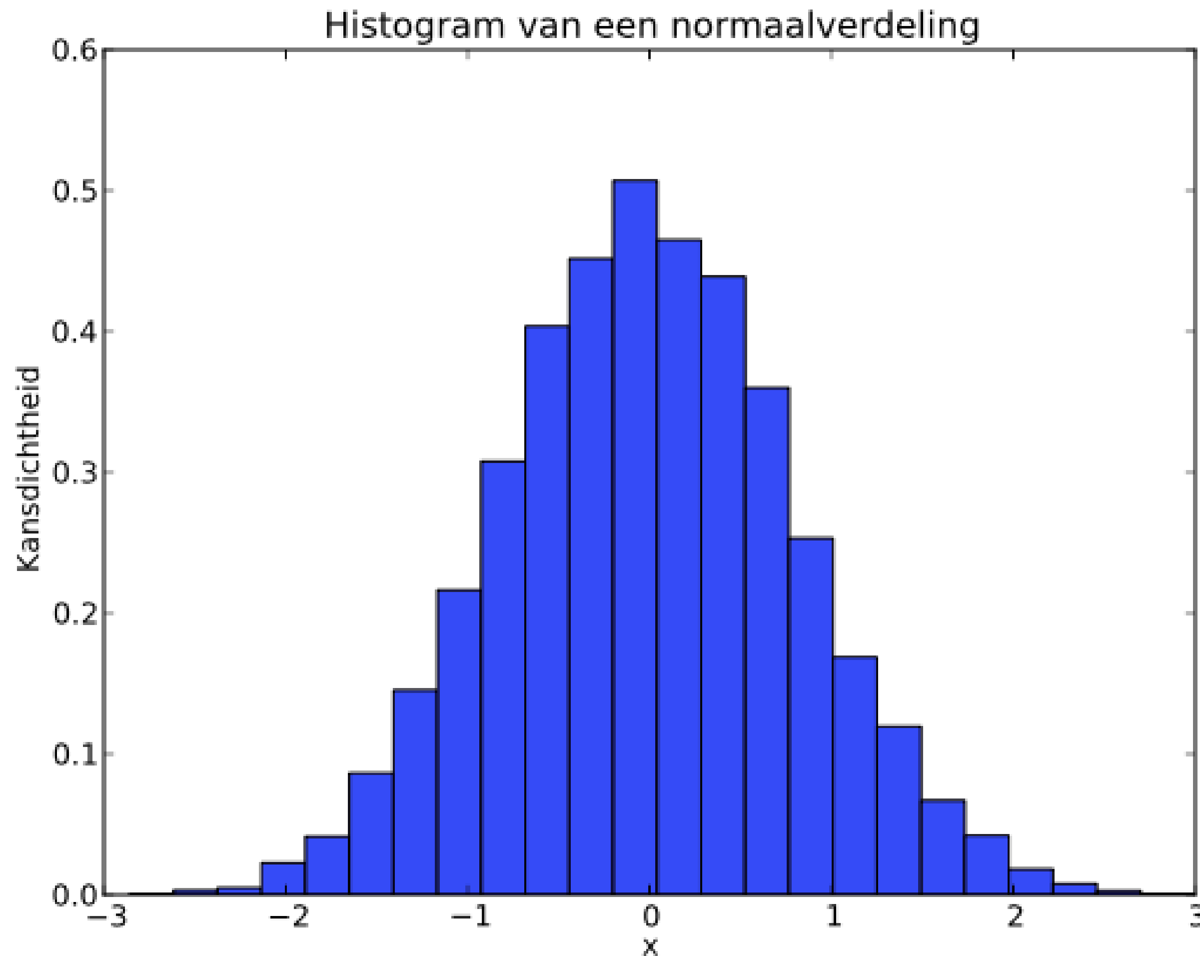


# Histogram maken

- NumPy kent ook vele kansverdelingen, zoals de normaalverdeling. We kunnen hier een histogramplot van maken.
- Je kunt ook zelf histogrammen maken van een gegeven array met behulp van `np.histogram( )`.

```
import numpy as np
import matplotlib.pyplot as plt

s = np.random.normal(0.0, 0.8, 1000)

# Histogram met 25 "bins" en normaliseer het histogram
plt.hist(s, bins=25, normed=True)

plt.title("Histogram van een normaalverdeling")
plt.xlabel("x")
plt.xlim(-3, 3)
plt.ylabel("Kansdichtheid")

plt.savefig("histogram.pdf")
```

# Histogram maken (2)



# Meerdere plots uit 1 array

- Het is mogelijk om met 1 `plt.plot()` aanroep meerdere reeksen te plotten.

- Dus bijvoorbeeld als je een meerdimensionale array hebt met meerdere reeksen, kun je deze in 1 keer plotten.

- Matplotlib zal elke *kolom* van de array als getallenreeks beschouwen.

- Een nadeel: we kunnen geen gebruik meer maken van de `label=` functionaliteit.
    - In plaats daarvan moeten we "handles" opvangen en doorgeven aan de legenda functie.

# Voorbeeld

```
x = np.linspace(-10, 10, 200)
y = np.zeros( (x.shape[0], 3) )
# Plaats 1 getallenreeks per kolom
y[:,0] = x ** 2
y[:,1] = 2 * x ** 2
y[:,2] = 4 * x ** 2

# x 1-d, y 2-d.
h = plt.plot(x, y)

# "h" bevat de handles.
plt.legend(h, ("1", "2", "3"))

plt.show()
```

# "Putting everything together"

- Veel programma's zullen bestaan uit:
  - Input: tekstbestand, CSV, gegenereerd, random, iets anders ...
  - Processing: rekenen mbv NumPy
  - Output: tekstbestand, CSV, binary array data of een plot.
- Voor Input en Output zul je waarschijnlijk heel vaak een Python module kunnen vinden die het meeste werk al kan klaren.

# Terug naar NumPy

# Reductieoperatoren langs een as

- Tot nu toe reductieoperatoren op volledige array. Uitkomst: een enkel element.
- Je kunt een reductie ook langs een bepaalde as van de array laten plaatsvinden.
- Geef als parameter mee: `axis=1` met `1` het nummer van de as (geteld vanaf 0).

# Reductie langs een as

```
>>> A = np.tile( [1,2,3], (3,1))
>>> A
array([[1, 2, 3],
       [1, 2, 3],
       [1, 2, 3]])
>>> A.sum(axis=0)
array([3, 6, 9])
>>> A.sum(axis=1)
array([6, 6, 6])
```

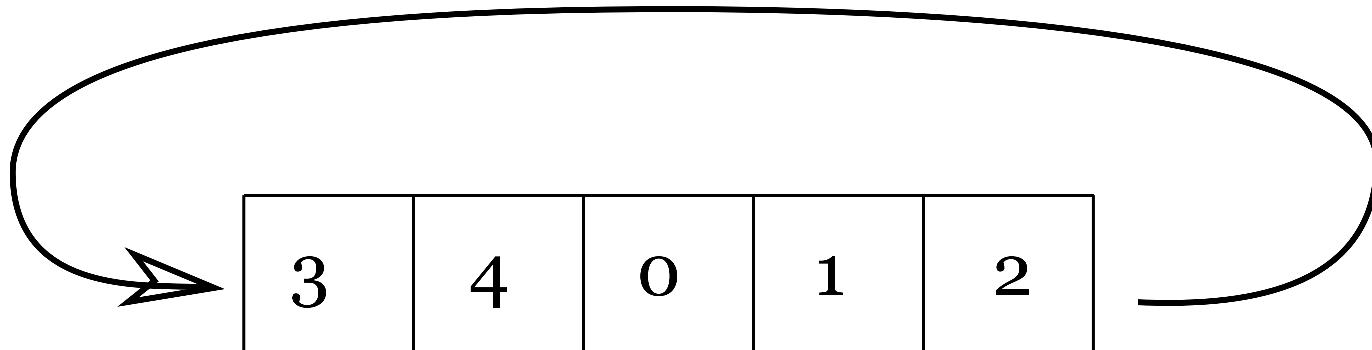
# `cumsum` operator

Met `cumsum` kun je cumulatief sommeren. Het aantal dimensies neemt dan niet af, dus dit is geen reductie operator.

```
>>> A = np.arange(15).reshape(3, 5)
>>> A
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
>>> A.cumsum()
array([  0,   1,   3,   6,  10,  15,
        21,  28,  36,  45,  55,  66,
        78, 91, 105])
>>> A.cumsum(axis=1)
array([[ 0,  1,  3,  6, 10],
       [ 5, 11, 18, 26, 35],
       [10, 21, 33, 46, 60]])
```

# Rollen

Door middel van "rollen" kunnen we elementen in een matrix "n" plaatsen opschuiven.




```
np.roll(np.arange(5), 2)
```

# Roteren en spiegelen

Tenslotte nog operaties om te roteren en te spiegelen.

```
>>> A = A.arange(0, 9).reshape( (3, 3) )
# Draai 90 graden tegen de klok in
>>> np.rot90(A)
array([[2, 5, 8],
       [1, 4, 7],
       [0, 3, 6]])
# Horizontaal spiegelen. Er is ook np.flipud()
>>> np.fliplr(A)
array([[2, 1, 0],
       [5, 4, 3],
       [8, 7, 6]])
```

# All en any

Om te kijken of een array aan een bepaalde conditie (Boolean expessie) voldoet, kunnen we gebruik maken van `np.all( )` en `np.any( )`.

```
>>> A = np.arange(10, 19).reshape( (3, 3) )
# Zijn alle elementen >= 15?
>>> np.all(A >= 15)
False
# >= 10?
>>> np.all(A >= 10)
True
# Is er tenminste een element gelijk aan 14?
>>> np.any(A == 14)
True
# En aan 4?
>>> np.any(A == 4)
False
# Tenminste een element kleiner dan 10?
>>> np.any(A < 10)
False
```

# Maskers

In de vorige slide gebeuren er eigenlijk twee dingen:

- Er wordt een Boolean array gemaakt: een masker.
- Er wordt gekeken of ten minste een, of alle, Boolean waarden True zijn.

```
>>> A = np.arange(10, 19).reshape((3, 3))
>>> A >= 15
array([[False, False, False],
       [False, False,  True],
       [ True,  True,  True]], dtype=bool)
# Tel aantal keer true in de Boolean array
>>> np.sum(A >= 15)
4
```

# Selectie van elementen

We kunnen maskers ook gebruiken om elementen te selecteren!

```
>>> mask = A >= 15
# Druk elementen >= 15 af. (Let op: 1-d view)
>>> print A[mask]
[15 16 17 18]
# Tel 100 op bij elementen >= 15
>>> A[mask] += 100
>>> print A
[[ 10  11  12]
 [ 13  14 115]
 [116 117 118]]
```

# Selectie van elementen (2)

Slice zowel rij als kolom gecombineerd:

```
>>> A = np.zeros( (5, 5) )
>>> m = np.zeros(A.shape, dtype=np.bool8)
>>> m[2,:] = True
>>> m[:,2] = True
>>> A[m] = 999
>>> A
array([[   0.,    0.,  999.,    0.,    0.],
       [   0.,    0.,  999.,    0.,    0.],
       [ 999.,  999.,  999.,  999.,  999.],
       [   0.,    0.,  999.,    0.,    0.],
       [   0.,    0.,  999.,    0.,    0.]])
>>> A[m]
array([ 999.,  999.,  999.,  999.,  999.,
999.,  999.,  999.,  999.])
```

# Lezen uit bestanden

```
f = open("getallen.txt", "r")
for line in f:
    line = line.rstrip("\n")
    a, b, c = line.split(" ")
    a, b, c = int(a), int(b), int(c)
    som = a + b + c
    print "Som:", som
f.close()
```

# Lezen uit bestanden (2)

Stel we willen alleen de eerste kolom opslaan in een NumPy array.

```
import numpy as np

f = open("getallen.txt", "r")
v1 = []
for line in f:
    line = line.rstrip("\n")
    a, b, c = line.split(" ")
    v1.append(int(a))
f.close()

A1 = np.array(v1)
print A1
print np.sum(A1 * 4)
```

# loadtxt() functie

NumPy heeft een ingebouwde functie om data uit tekstbestanden te laden die bestaan uit regels met getallen.

```
# gegeven het volgende bestand
1 2 3
4 5 6
0 9 1
9 3 4
# code:
>>> A = np.loadtxt("test1.txt")
>>> print A
[[ 1.  2.  3.]
 [ 4.  5.  6.]
 [ 0.  9.  1.]
 [ 9.  3.  4.]]
```

# `loadtxt()` functie (2)

- Getallen gescheiden door komma's:

```
A = np.loadtxt("test2.txt", delimiter=",")
```

- Strings inladen gaat niet vanzelf goed (tenzij je zelf het datatype goed instelt). Maak gebruik van `skiprows` en `usecols` om rijen/kolommen met strings over te slaan.

- `np.genfromtxt()` is een zelfde soort functie met meer opties.

# Driehoek van Pascal

```
            1
          1    1
       1    2     1
    1    3     3    1
  1    4    6     4    1
1   5   10     10    5     1
```

```
1
1 1          A[i-1,j-1] + A[i-1,j]
1 2 1             = A[i,j]
1 3 3 1
1 4 6 4 1
1 5 10 10 10 5 1
```

# Driehoek van Pascal (2)

```
def pascaldriehoek(n):
    pascal = np.zeros( (n, n), dtype=np.int32 )
    # Nulde kolom bevat enen
    pascal[:,0] = 1
    pascal[0,1] = 0
    print pascal[0,0],
    for i in range(1, n):
        print "\n", pascal[i,0],
        for j in range(1, i+1):
            pascal[i,j] = pascal[i-1,j-1] + pascal[i-1,j]
            print pascal[i,j],
        if i != n - 1:
            pascal[i,i+1] = 0
```

# Driehoek van Pascal (3)

In de functie wordt de hele driehoek (tijdelijk) opgeslagen in een 2-dimensionale array `pascal`. Dit kan efficienter!

We kunnen volstaan met een 1-dimensionale array `rij`, dat telkens de `i`-de rij uit de driehoek van Pascal bevat. Om de `(i+1)`-de rij te vinden gebruiker we immers alleen de `i`-de rij, dus alle vorige rijen zijn niet meer nodig.

Merk op dat een volgende rij nu altijd van rechts naar links gevuld moet worden, omdat je anders waarden overschrijft die je nog nodig hebt.

# Driehoek van Pascal (4)

```
def pascaldriehoekbeter(n):
    rij = np.zeros(n, dtype=np.int32)
    # 0-de kolom altijd 1
    rij[0] = 1
    print rij[0]
    for i in range(1, n):
        for j in reversed(range(1, i+1)):
            rij[j] = rij[j-1] + rij[j]
            print rij[j],
        print rij[0]
```

# Insertion sort

```
def invoegsorteer(A):
    N = A.shape[0]
    # We zetten steeds A[i] goed in reeds
    # gesorteerde beginstuk
    for i in range(1, N):
        temp = A[i]
        j = i - 1
        while j >= 0 and A[j] > temp:
            A[j+1] = A[j]
            j -= 1
        A[j+1] = temp
```

# Matrix optelling

***Gebruik altijd de NumPy operator om matrices op te tellen!***

Maar als voorbeeld, hoe zouden we zelf matrix optelling moeten implementeren?

```
def matadd(A, B):
    # Stel zeker dat A en B dezelfde dimensies hebben
    assert A.shape == B.shape

    C = np.zeros(A.shape)
    for i in range(0, A.shape[0]):
        for j in range(0, A.shape[1]):
            C[i,j] = A[i,j] + B[i,j]
    return C
```

# Matrix vermenigvuldiging

***Gebruik altijd de NumPy operator om matrices te vermenigvuldiging!***

Maar als voorbeeld, hoe zouden we zelf matrix vermenigvuldiging moeten implementeren?

```
def matmul(A, B):
    # Stel zeker dat kolomdim. A overeenkomt met rijdim. B
    assert A.shape[1] == B.shape[0]

    M, K, N = A.shape[0], A.shape[1], B.shape[1]

    C = np.zeros( (M, N) )
    for i in range(0, M):
        for j in range(0, N):
            for k in range(0, K):
                C[i,j] += A[i,k] * B[k,j]
    return C
```

# Zeef van Eratosthenes

```
N = 1000
wortel = np.sqrt(N)
# Initialiseer op True, tot tegendeel bewezen is ...
zeef = np.ones(N, dtype=np.bool8)
zeef[0] = False
zeef[1] = False
for getal in range(2, int(wortel)):
    if zeef[getal]:
        # Streep veelvouden door
        veelvoud = 2 * getal
        while veelvoud < N:
            zeef[veelvoud] = False
            veelvoud += getal
for getal in range(2, N):
    if zeef[getal] == True:
        print getal,
# Of: print np.where(zeef == True)
```

# Tot slot

- Werkcollege:
  - Op papier II
  - En verder met de programmeeropdracht.

- Volgende week:
  - Restje NumPy (3-dimensionale arrays)
  - iPython
  - Python module showcase