Python voor Natuur- en Sterrenkundigen Week 4

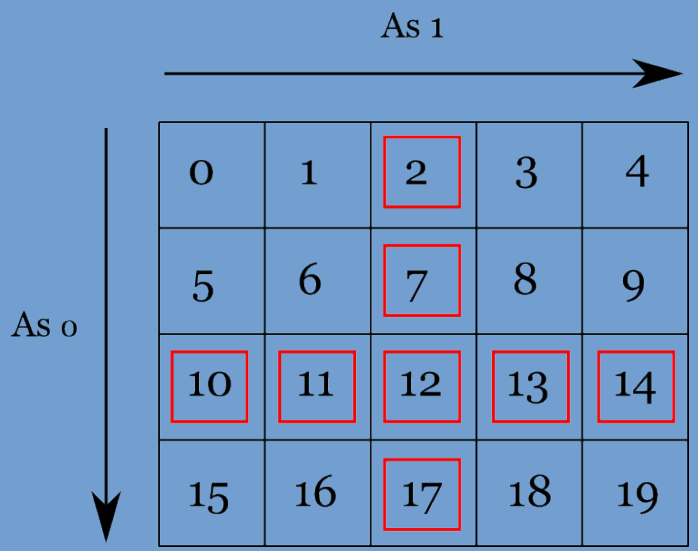
Kristian Rietveld

http://liacs.leidenuniv.nl/~rietveldkfd/courses/pmpy2015/

Universiteit Leiden
The Netherlands

Vorige week

- Geparkeerde vraag: hoe maken we een slice van zowel een rij als kolom gecombineerd?



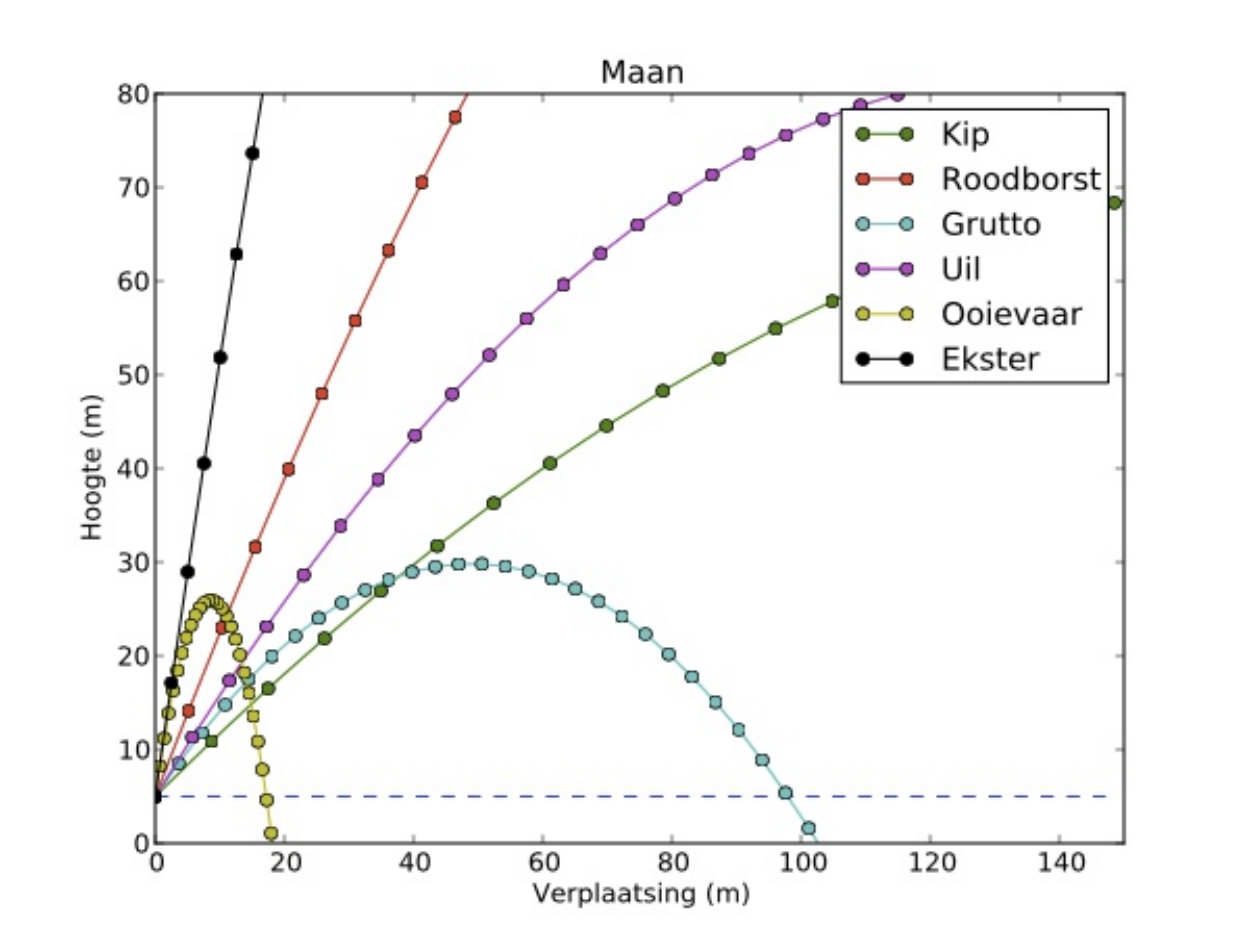
- Gerelateerd: hoe maken van een slice van een diagonaal?

Eindopdracht

- Het is de bedoeling dat je zelf de bestanden inleest met behulp van `open`. Zie ook het dictaat voor voorbeelden!

Eindopdracht (verv.)

- Banen van de maan.



Eindopdracht (verv.)

- Wat moet er in het verslag?
  - Beschrijving programma / probleem.
  - Plaats de gemaakte plots in je verslag. (graphicx package)
  - Omschrijf iets dat je opviel in de plots.
  - Tabel met de gewerkte uren (per persoon per week).
  - Python code via LaTeX listings, gebruik `language=Python` voor correcte "syntax highlighting".

Floating-point getallen

- Reele getallen worden in computers opgeslagen als "floating-point" getal.
- De opslagcapaciteit is niet onbeperkt, dus we kunnen niet alle mogelijke getallen opslaan.
- Er vindt afronding plaats!
- Irrationele getallen als π, $\sqrt{(2)}$ worden in ieder geval nooit exact gepresenteerd!

Floating-point getallen (verv.)

- IEEE 754 standaard.
- Python: double precision, 64 bits.
  - 1 bit sign
  - 11 bits exponent
  - 53 bits significant
- $1.11011 * 2^{-3} => 0.23046875$
- 53 bits => 16 significante cijfers.
- Bereik: -10^{308} tot 10^{308}.

Floating-point getallen (verv.)

- Wanneer opletten?
  - Er zullen afrondfouten optreden, voornamelijk bij iteratieve berekeningen.
  - Verschillende volgorde in toepassen operatoren: verschillende resultaten.
  - Operatie op een zeer klein en zeer groot getal: significantie kleine getal verdwijnt.
  - Deling van integer getallen geeft standaard een integer.
  - Pas op met vergelijken van floating-point getallen! 6.1 kan zijn opgeslagen als 6.099999999999996. Dan doet `x == 6.1` wellicht iets anders.
    - Oplossing: maak gebruik van `np.allclose(a, b)`.

Arrays met meerdere dimensies

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

```
>>> A = np.tile(6, (3, 4))
>>> print A
[[6 6 6 6]
 [6 6 6 6]
 [6 6 6 6]]
```

Slicing

```
A[3,4]

A[3,:]

A[3:5,1:3]
```

Slicing (verv.)

```
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)
```

Diagonaal aanpassen (verv.)

```
>>> 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.]])
```

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]]
```

Rekenen in meerdere dimensies

```
>>> np.ones( (3, 3) ) + np.tile(10, (3, 3) )
array([[ 11., 11., 11.],
       [ 11., 11., 11.],
       [ 11., 11., 11.]])
```

Rekenen in meerdere dimensies (verv.)

```
>>> np.eye(4) * 9
array([[ 9., 0., 0., 0.],
       [ 0., 9., 0., 0.],
       [ 0., 0., 9., 0.],
       [ 0., 0., 0., 9.]])
```

Rekenen in meerdere dimensies (verv.)

```
>>> np.arange(9).reshape( (3, 3) ) * 2
array([[ 0,  2,  4],
       [ 6,  8, 10],
       [12, 14, 16]])
```

* is geen dot product!

- Wat gebeurt hier???

```
>>> A = np.eye(3)
>>> B = np.tile(4, (3, 3))
>>> A * B
array([[ 4.,  0.,  0.],
       [ 0.,  4.,  0.],
       [ 0.,  0.,  4.]])
```

Matrix/dot product

- Gebruik `np.dot()` als je een matrix/dot product wilt doen.

```
>>> np.dot(A, B)
array([[ 4.,  4.,  4.],
       [ 4.,  4.,  4.],
       [ 4.,  4.,  4.]])
```

- Of maak matrices met `np.matrix()` ipv `np.array()`.

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

- Shape (3,): `[1, 2, 3]`
- Shape (1, 3): `[[1, 2, 3]]`
- Let op de extra blokhaak corresponderend met de extra dimensie.

Standaardgedrag

- NumPy probeert overigens 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`).

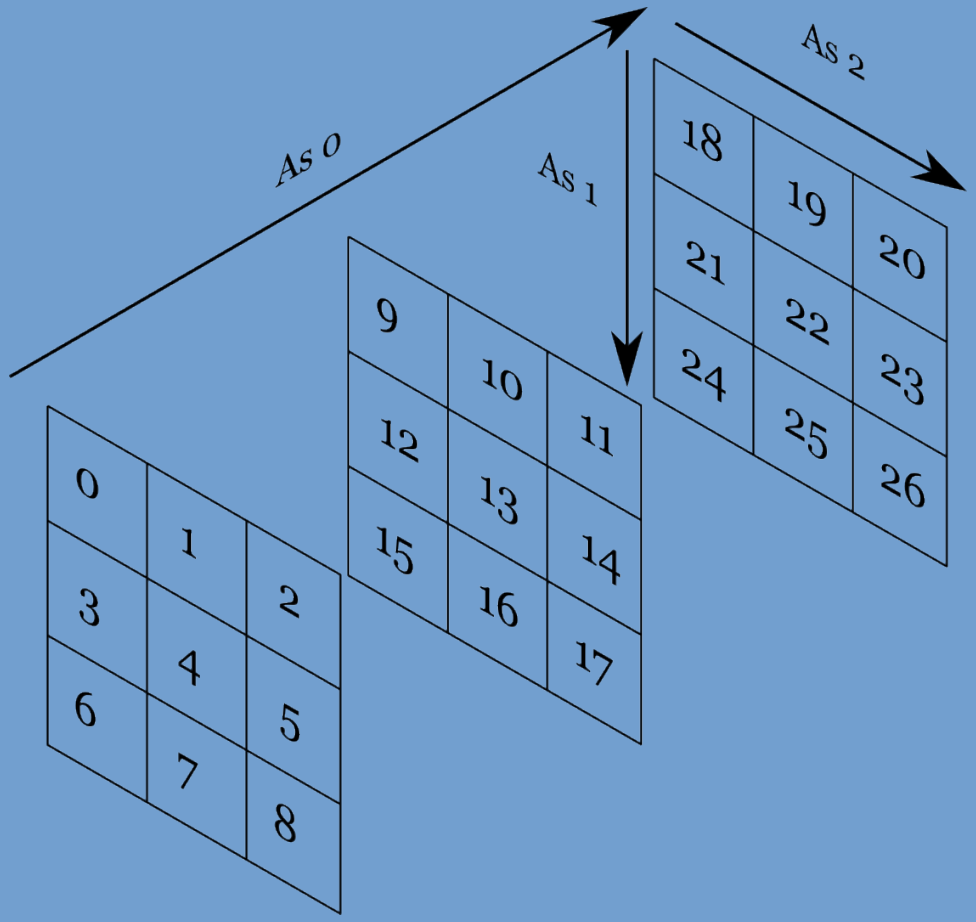
Gebruik kolomvectoren (verv.)

```
>>> 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.]])
```

3-dimensionale arrays

- Ook 3-dimensies is helemaal geen probleem.
- Initialisatie zoals je bent gewend.
- Vorm-tuple bevat 3 waarden.

3-dimensionale arrays



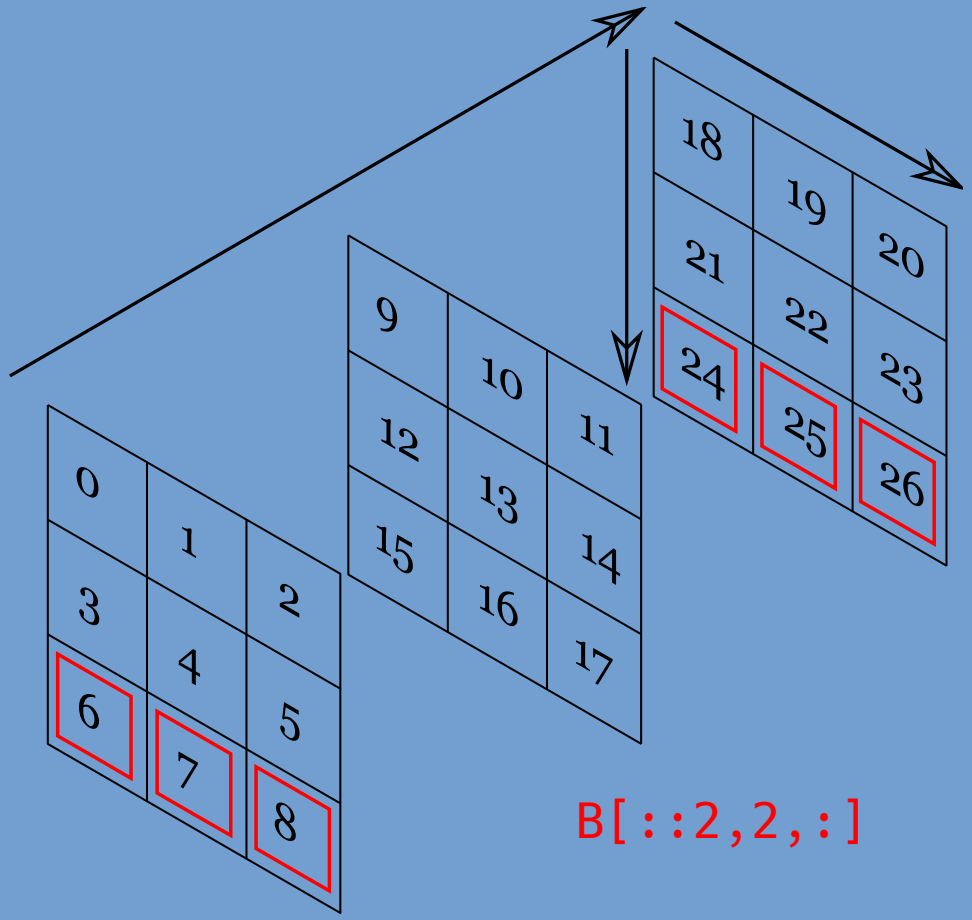
3-dimensionale arrays (verv.)

- Drie dimensies, heeft dat nu wel zin?
- Voorbeeld:
  - 2 vlakken: 1 voor x-coordinaten, 1 voor y-coordinaten.
  - Per vlak: N tijdstappen langs de rij-as.
  - Per vlak: M verschillende vogels langs de kolom-as.
  - (2, N, M)

Slicing

```
>>> B = np.arange(27).reshape( (3,3,3) )
# Kies alleen "voorste" vlak; B[0] is equivalent.
>>> B[0,:,:]
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
# Kies uit het voorste vlak de derde kolom.
>>> B[0,:,2]
array([2, 5, 8])
# Uit vlakken 0, 2, ... kies de derde rij.
>>> B[::2,2,:]
array([[ 6,  7,  8],
       [24, 25, 26]])
```

Slicing (verv.)




```
B[::2,2,:]
```

Slicing (verv.)

```
# Uit alle vlakken, selecteer rij/kolom 0, 2, ...
>> B[:,::2,::2]
array([[[ 0,  2],
        [ 6,  8]],

       [[ 9, 11],
        [15, 17]],

       [[18, 20],
        [24, 26]]])
```

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=2` met 2 het nummer van de as (geteld vanaf 0).

Reductie langs een as

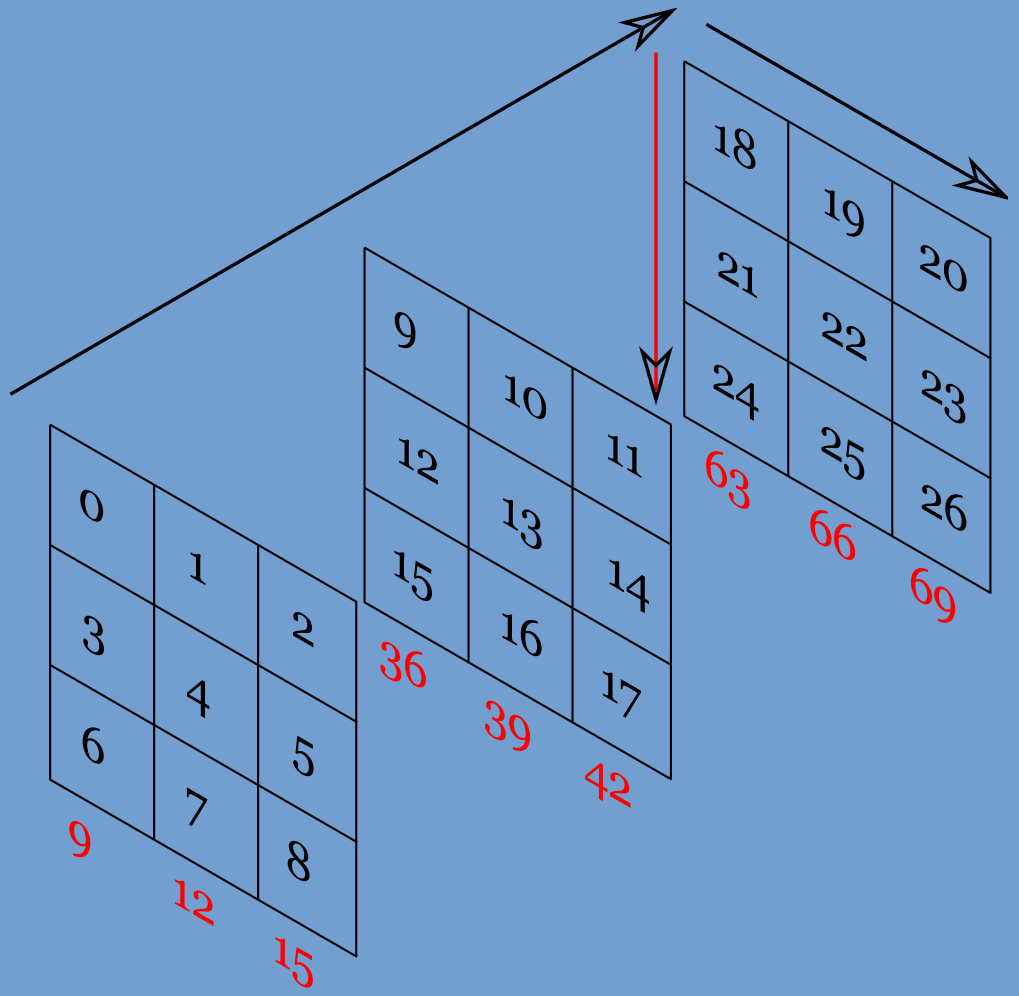
```
>>> A = np.tile( [1,2,3], (3,1))
>>> A
array([[1, 2, 3],
       [1, 2, 3],
       [1, 2, 3]])
>>> A.sum(axis=0)
array([3, 6, 9])
>>> A.sum(axis=1)
array([6, 6, 6])
```

Reductie langs een as (verv.)

- Of 3-dimensionaal.

```
>>> B = np.arange(27).reshape( (3,3,3) )
# Sommeer elke kolom (dus langs de rij-as)
>>> B.sum(axis=1)
array([[ 9, 12, 15],
       [36, 39, 42],
       [63, 66, 69]])
```

Reductie langs een as (verv.)



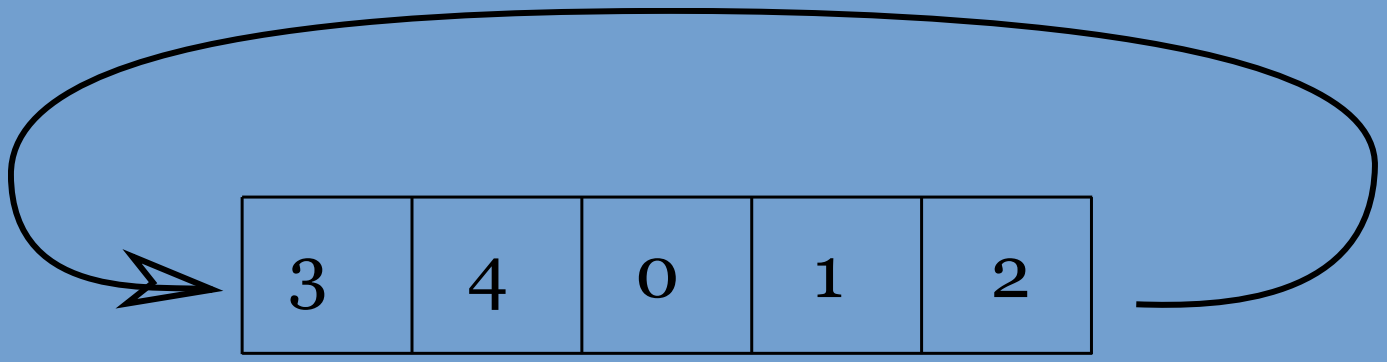
cumsum operator

- 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.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 elmenten >= 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]]
```

Geparkeerde vraag

- Slice zowel rij als kolom gecombineerd.

```
>>> A = np.zeros( (5, 5) )
>>> m = np.zeros(A.shape, dtype=np.bool_)
>>> 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.])
```

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

- 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.

CSV module

- CSV: Comma Separated Values.
- Wordt ondersteund door elke spreadsheet.
- Python module voor inlezen/wegschrijven.

CSV module (verv.)

```
import csv
f = open("data.csv", "r")
csvreader = csv.reader(f)
for row in reader:
    # elke row is een Python list
    print row
f.close()
```

Random Numbers

- Met `np.random.random()` kun je zowel random getallen als arrays maken. De elementen zullen zitten in het interval [0.0, 1.0).

```
>>> np.random.random()
0.9420733512975746
>>> np.random.random( (3, 3) )
array([[ 0.85083159,  0.28587965,  0.69833045],
       [ 0.98522151,  0.93762675,  0.29451167],
       [ 0.17332978,  0.87714118,  0.36772117]])
```

Random Numbers (verv.)

- Voor een integer range, maak gebruik van `np.random.random_integers()`:

```
# Integers tussen 0 t/m 10, shape (3, 3)
>>> np.random.random_integers(0, 10, (3, 3) )
array([[6, 9, 4],
       [8, 0, 5],
       [8, 7, 1]])
```

Random Numbers (verv.)

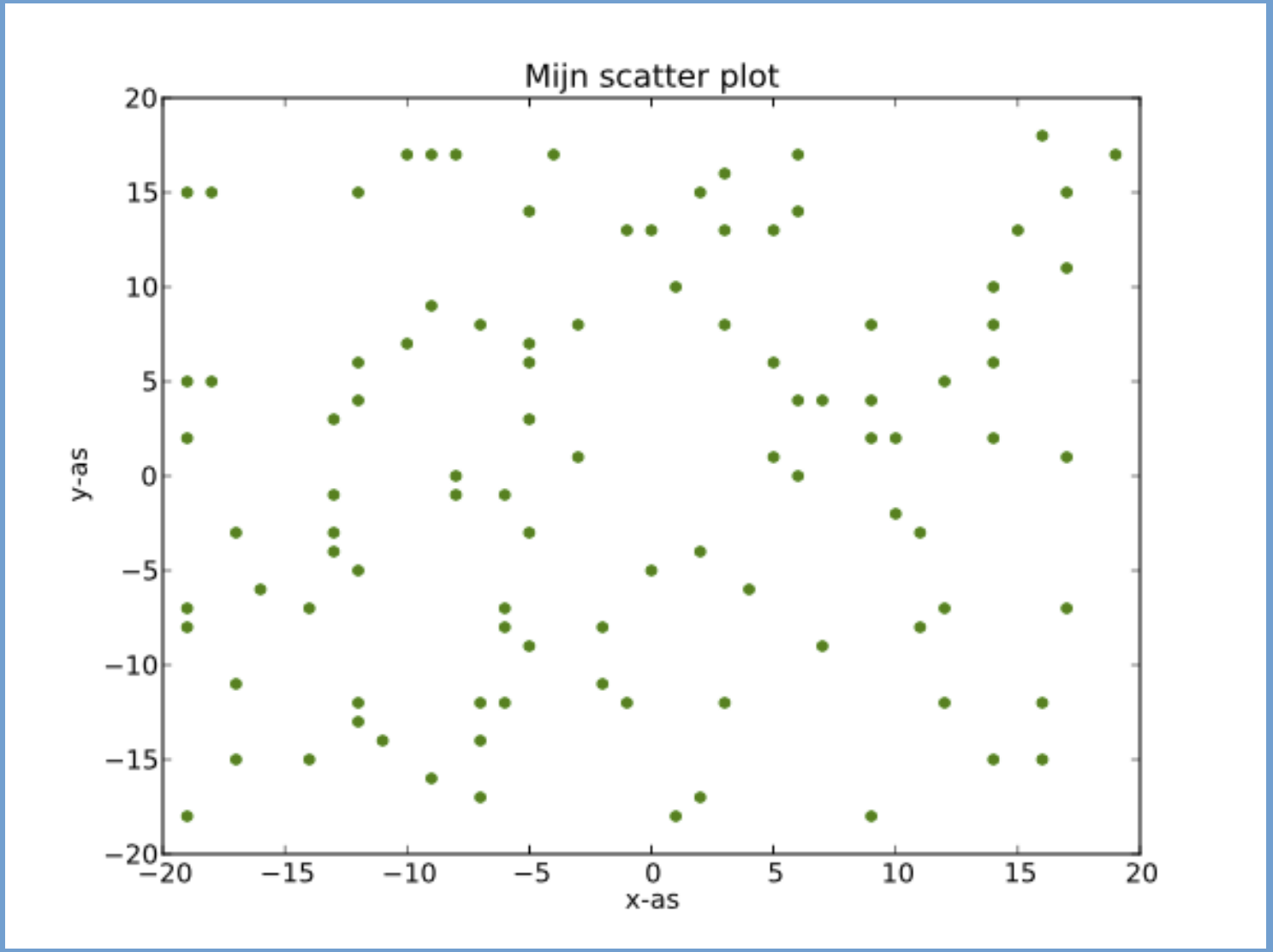
- Wat ook vaak voorkomt is dat je een trekking wilt doen uit een reeks van getallen.
- Standaard gebeurt dit "met terugleggen". Voor zonder terugleggen voeg toe `replace=False`.

```
>>> np.random.choice(np.arange(100, 200))
117


# Trekking van 5 elementen
>>> np.random.choice(np.arange(100, 200), 5)
array([190, 135, 176, 112, 101])
```

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")

exit(0)
```

Scatter plots



Histogram maken

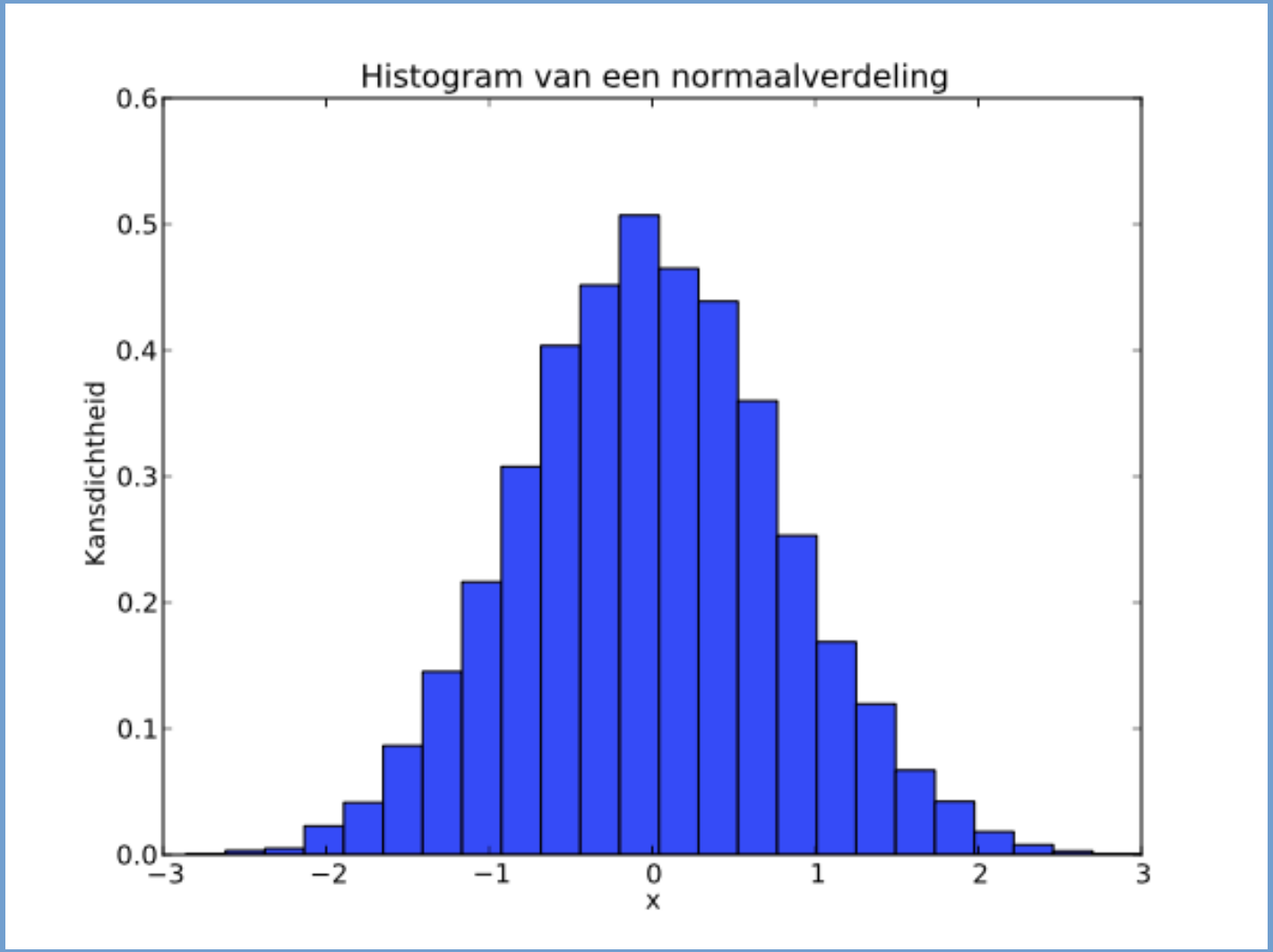
- NumPy kent ook vele kansverdelingen, zoals de normaalverdeling. We kunnen hier een histogramplot van maken (volgende slide).

- Je kunt ook zelf histogrammen maken van een gegeven array met behulp van `np.histogram()`.

Histogram maken (verv.)

```
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")

exit(0)
```

Histogram maken (verv.)



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 = 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.

En nu?

- Er zijn veel meer mogelijkheden binnen de taal Python: kort overzicht.
- En er zijn vele handige Python modules en packages die je kunt leren gebruiken!

Meer Python

- Python is volledig object-georienteerd: klassen, operator overloading.
- Daarnaast: generators, list comprehensions, lambda functies, exceptions, ...
- Python Tutorial: https://docs.python.org/2/tutorial/

Meer Python (OOP)

```
class MijnKlasse(object):
    def __init__(self, naam):
        self.naam = naam

    def zeg_hallo(self):
        print self.naam + " zegt hallo"

test = MijnKlasse("Testpersoon")
test.zeg_hallo()
```

Meer Python (generators)

```
def graaf_tel():
    reeks = range(1, 7)
    for getal in reeks:
        yield getal

for i in graaf_tel():
    print i
```

Meer Python (list comprehensions)

```
x = [0 for i in range(10)]
x = [i for i in A if i < 10]

strings = map(str, [i for i in graaf_tel()])

som = sum(i for i in graaf_tel())
```

Module showcase

- De Python bibliotheek is al zeer uitgebreid.

- Documentatie online:
  https://docs.python.org/2/library/index.html

Module documentatie

- De documentatie is meestal als volgt gestructureerd:
  - Omschrijving inhoud en doel module.
  - Omschrijving alle klassen en functies in de module. Uitleg werking en parameters functies.
  - Aan het einde vind je vaak enkele voorbeelden.

Modules uit de standaardbiliotheek

- re: regular expressions
- datetime & calendar
- decimal & fraction
- zipfile & tarfile
- SQL DB toegang
- Internet modules: e-mail, HTTP, FTP, ...
- UNIX / Mac / Windows specifieke modules
- En nog veel meer ...

Externe Packages

- Korte showcase van interessante packages.
- Deze slides zijn gegenereerd met een Python script!!
- Hoe installeren?
  - Linux: liefst via Linux distributie, anders "pip".
  - Mac: of MacPorts, of Python distributie, of "pip".
  - Windows: via Python distributie.
  - (Zie ook het dictaat voor links)

Packages zoeken

- PyPI: Python Package Index.
  - https://pypi.python.org/pypi
- Of Google …

SciPy

- Meer science & mathematics functionaliteiten.
- Constanten (natuurkunde/sterrenkunde).
- I/O: MATLAB matrices, IDL, wave files, sparse matrices.
- Lineaire algebra.
- Fourier Transforms.
- Integratie & differentiaal vergelijkingen.
- Etc…

Astropy

- Astronomische coordinaten
- Model fitting
- Convolution
- Cosmologische modellen …

Andere natuurkundige modules

- ElectromagneticPython
- gwpy - gravitational wave astrophysics
- PyFeyn - Feynman diagrammen tekenen
- SunPy - Solar Physics
- ...

Pandas

- Data manipulatie & analyse.
- Lijkt op meer op een "spreadsheet" vergeleken met NumPy.
- Kan direct CSV inlezen, begrijpt headers.
- Tegenhanger van "R".

Excel

- Verschillende modules om te werken met Excel files:
  - xlrd
  - xlsxwriter
  - xlutils
- Of Excel spreadsheets manipuleren vanuit Python!
  - xlwings

Hoe nu verder?

- Veel gebruik maken van Python is de beste manier om het goed onder de knie te krijgen!

- Veel succes & plezier!