Python voor Natuur- en Sterrenkundigen Week 3

Kristian Rietveld

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

Universiteit Leiden
The Netherlands

Deze week

- Voortbouwen op onze basiskennis Python
- Modules & Packages
- NumPy
- Matplotlib

Modules en packages

- Tot nu toe alleen programma's geschreven die bestonden uit een enkel bestand.
- Code kun je verspreiden over meerdere `.py`-bestanden.
- Hoe roepen we een functie uit een ander `.py`-bestand aan?

`import` statement

- Een functie moet gedefinieerd zijn, voordat je deze in Python kunt aanroepen.
- Functies uit andere bestanden eerst *importeren*.
- We noemen dit soort andere bestanden "modules".
- Bundeling van modules: "package".

import statement (verv.)

```
# importeer de gehele module, let op we laten ".py" weg!
import handig

handig.hallo()
c = handig.telop(a, b)
c = handig.vermenigvuldig(a, b)
```

import statement (verv.)

```
# importeer een specifieke functie uit een module
from handig import telop

# We hoeven nu niet de prefix "handig." te gebruiken
c = telop(a, b)
```

import statement (verv.)

```
# importeer de gehele module, maar onder een afgekorte naam
import handig as h

h.hallo()
c = h.telop(a, b)
```

Zelf modules maken

- Hoe maken we nu zelf een module?
- Maak een aparte `.py`-bestand met daarin functies.
  - Let op: gebruik geen streepjes of spaties in de bestandsnaam!
- Importeer het bestand met `import`.

Voorbeeld

```
def hallo():
    print "hello world"

def telop(a, b):
    return a + b

def vermenigvuldig(a, b):
    return a * b
```

NumPy introductie

- NumPy: Numerical Python.
- Wordt in heel veel takken van de wetenschap gebruikt voor numeriek rekenwerk.
- Belangrijkste onderdeel: multidimensionale array datastructuur.

NumPy import

- NumPy is een package en moeten we eerst importeren.

```
import numpy as np
```

Intermezzo: iPython

- De interactieve Python prompt is handig, maar het kan nog veel beter.
- iPython: "turbocharged" interactief Python.
- Ideaal in combinatie met NumPy en matplotlib.

iPython features

- Je kan makkelijk voorgaande resultaten hergebruiken.
- Je kan ook ls, cat, cd, etc. gebruiken.
- Tab completion (!)
- Pylab mode.

De NumPy array

- Multidimensionale array zoals je ook in C++ hebt leren kennen.
- Aantal belangrijke verschillen ten opzichte van Python lijsten:
  - Aantal elementen staat na aanmaken vast.
  - Alle elementen zijn van hetzelfde type.
  - Gebruik van operatoren op NumPy arrays is wat je zou verwachten in tegenstelling tot Python lijsten (zie ook later).

NumPy arrays maken

- We beginnen met 1-dimensionale arrays.
- Bij het maken geven we het aantal elementen op.
- Verschillende manieren:
  - Creeren aan de hand van een Python list.
  - `np.zeros`: initialisatie met nullen.
  - `np.ones`: initialisatie met nullen.
  - `np.tile`: initialisatie met gespecificeerde waarde.

NumPy arrays maken (verv.)

```
>>> np.array([1, 2, 3, 4, 5, 6])
array([1, 2, 3, 4, 5, 6])
>>> np.zeros(6)
array([ 0.,  0.,  0.,  0.,  0.,  0.])
>>> np.ones(6)
array([ 1.,  1.,  1.,  1.,  1.,  1.])
>>> np.tile(39., 6)
array([ 39.,  39.,  39.,  39.,  39.,  39.])
```

NumPy arrays maken (verv.)

- `np.arange(start, stop, stap)`: maak een getallen reeks. Mag ook floating-point gebruiken!

- `np.linspace(begin, eind, N)`: N getallen uit gesloten interval, gelijke afstand tussen de elementen.

NumPy arrays maken (verv.)

```
>>> np.arange(0, 10, 2)
array([0, 2, 4, 6, 8])
>>> np.linspace(1, 5, 10)
array([ 1.        ,  1.44444444,  1.88888889,  2.33333333,
2.77777778,
        3.22222222,  3.66666667,  4.11111111,  4.55555556,  5.
   ])
```

Eigenschappen van NumPy arrays

```
>>> A = np.zeros(6)      # 6 elementen, waarde nul.
>>> A.ndim               # Aantal dimensies.
1
>>> A.shape      # De grootte van elke dimensie (zie ook later).
(6,)
>>> A.size       # Het aantal elementen in de array.
6
>>> A.dtype      # Het datatype van elk element (zie ook hieronder)
dtype('float64')
```

Datatypen in NumPy

- `float64`? Die hebben we nog niet eerder gezien.
- NumPy kent vele extra datatypen waaruit kan worden gekozen om de data zo efficient mogelijk op te slaan.
- De belangrijkste: `np.bool_`, `np.int32`, `np.float64`, `np.complex128`.

Datatypen in NumPy

- Bij initialisatie probeert NumPy een geschikt datatype te kiezen.
- Soms is de gok niet wat je wilt, zelf opgeven met `dtype=`.

```
>>> np.ones(10)
array([ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
1.])
>>> np.ones(10, dtype=np.int32)
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1], dtype=int32)
```

Lijst vs. NumPy array

- Laten we eens gaan rekenen met een lijst.

```
>>> l = [1, 2, 3, 4]
>>> l * 4
[1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4]
>>> l + 4
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate list (not "int") to list
>>> l * l
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can't multiply sequence by non-int of type
'list'
```

Rekenen met NumPy arrays

- Python lijsten geven ons niet de resultaten die we zouden verwachten.
- Daarom: als je gaat rekenen, gebruik NumPy arrays!
- Operatoren werken elementgewijs.

Rekenen met NumPy arrays (verv.)

```
>>> a = np.array([1, 2, 3, 4])
>>> a * 4
array([ 4,  8, 12, 16])
>>> a + 4
array([5, 6, 7, 8])
>>> a * a
array([ 1,  4,  9, 16])
```

Rekenen met NumPy arrays (verv.)

- Toepassen formule op een getallenreeks.

```
>>> x = np.arange(0, 10)
>>> print x
[0 1 2 3 4 5 6 7 8 9]
>>> f1 = x ** 2
>>> f1
array([ 0,  1,  4,  9, 16, 25, 36, 49, 64, 81])
>>> f2 = x ** 3 + 2 * x**2 - 3
>>> f2
array([ -3,   0,  13,  42,  93, 172, 285, 438,
637, 888])
```

Reductieoperatoren

- Een reductieoperator berekent 1 resultaat voor een gehele array.
- Voorbeelden:
  - Sommeren: `np.sum()`
  - Gemiddelde: `np.mean()`
  - Standaardeviatie: `np.std()`
  - Minimum: `np.amin()`
  - Maximum: `np.amax()`

Wiskundige functies

- Alle belangrijke wiskundige functies vind je terug in NumPy.
- Parameter mag natuurlijk zowel een scalair als array zijn.
- Voorbeelden:
  - `np.log()`, `np.log10()`, `np.exp()`
  - `np.sin()`, `np.cos()`, `np.tan()`
  - Let op: `np.deg2rad()`.
  - `np.sqrt()`, `np.floor()`, `np.ceil()`
- Constanten: `np.pi`, `np.e`.
- (Natuurkundige constanten: zie Scipy).

Slicing & indexing

- Indexing en slicing zoals je bent gewend.
- Toekenning aan een slice:
  - Toekenning scalair: elk element in de slice krijgt deze waarde.
  - Toekenning array: arrays moeten evenveel elementen bevatten!

Slicing & indexing (verv.)

```
>>> A = np.arange(0, 10)
>>> A[1:4] = 10
>>> print A
[ 0 10 10 10  4  5  6  7  8  9]
>>> A[8:] = [20, 21, 22, 23]    # Reeks om toe te
kennen groter dan slice
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: cannot copy sequence with size 4 to
array axis with dimension 2
>>> A[8:] = [20, 21]
>>> print A
[ 0 10 10 10  4  5  6  7 20 21]
```

Multidimensionale arrays

- NumPy arrays kunnen een arbitrair aantal dimensies aan.
- Dimensies worden ook wel "assen" genoemd.
- Elke as heeft ene bepaalde lengte.
- Elke NumPy array heeft een "vorm" waarin de lengte van elke as is vastgelegd.
  - `(3, )`: 1 dimensie lengte 3.
  - `(3, 4)`: 2 dimensies: 3 rijen, 4 kolommen.
  - `(10, 3, 4)`: 3 dimensies: 10 vlakken, 3 rijen, 4 kolommen. (volgende week).

Multidimensionale arrays (verv.)

- Om te maken werken de gebruikelijke functies. In plaats van een aantal elementen vul je een shape tuple in.

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

Identiteitsmatrices

- `np.eye(n)` maakt een n x n identiteitsmatrix.

```
>>> I = np.eye(3)    # Een 3x3 identiteitsmatrix
>>> print I
[[ 1.  0.  0.]
 [ 0.  1.  0.]
 [ 0.  0.  1.]]
```

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

Hoe zit dat met blokhaken?

- Het aantal blokhaken correspondeert met het aantal dimensies.

- `[0 1 0]` is iets anders dan `[[0 1 0]]`, zie ook volgende week.

Arrays kopieren

- Pas op: een toekenning is geen kopieeractie!!

```
>>> A = np.eye(3)
>>> B = A        # Kopieert niet, maar legt een extra
referentie aan.
>>> B[0,2] = 9   # Indexeren komen we later op
>>> print A      # A is dus ook aangepast!
[[ 1.  0.  9.]
 [ 0.  1.  0.]
 [ 0.  0.  1.]]
>>> B = np.copy(A)  # De correcte manier om een kopie te
maken.
```

Indexeren over meerdere dimensies

- Om een element aan te duiden in een multidimensionale array: geef per as (dimensie) een index op, gescheiden door komma's.
- `B[0,2]`
- `B[1,2,3,4,5]`

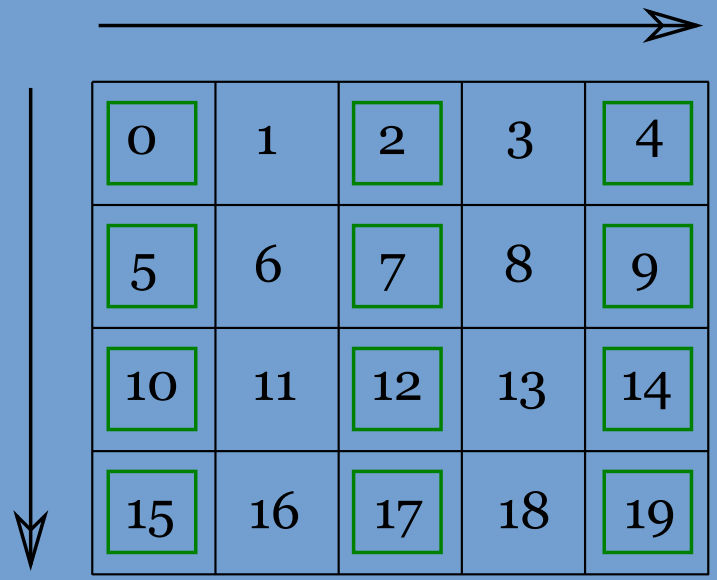
Slicing over meerdere dimensies

- In plaats van een index mag je natuurlijk ook een slice opgeven.
- De lege slice `:` selecteert de gehele as.

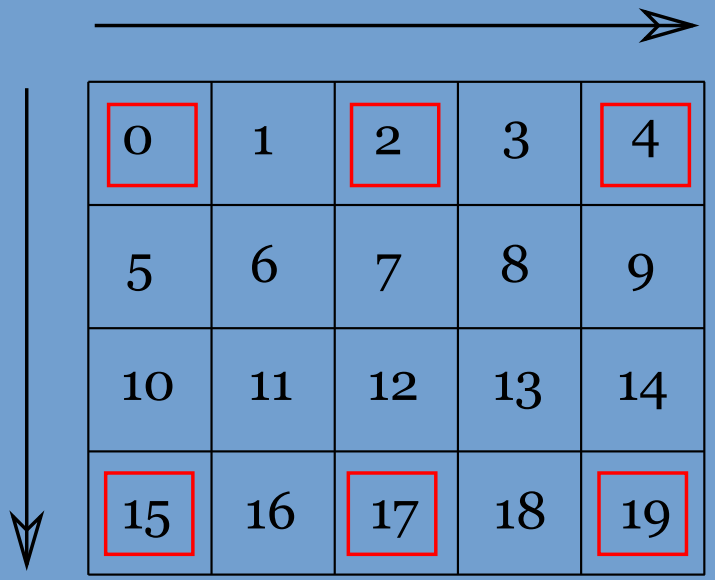
Slicing (verv.)

```
>>> A[:,:]
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19]])
>>> A[2,1]          # Selecteer een enkel element
11
>>> A[2,:]          # Selecteer de derde rij.
array([10, 11, 12, 13, 14])
>>> A[2]            # Slices aan het einde mag je weglaten
array([10, 11, 12, 13, 14])
>>> A[:,3]          # Selecteer de vierde kolom
array([ 3,  8, 13, 18])
```

Slicing (verv.)



`A[:,::2]`



`A[::3,::2]`

Volgende week meer!

Maar nu eerst: plotten.

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 plot.
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()

exit(0)
```

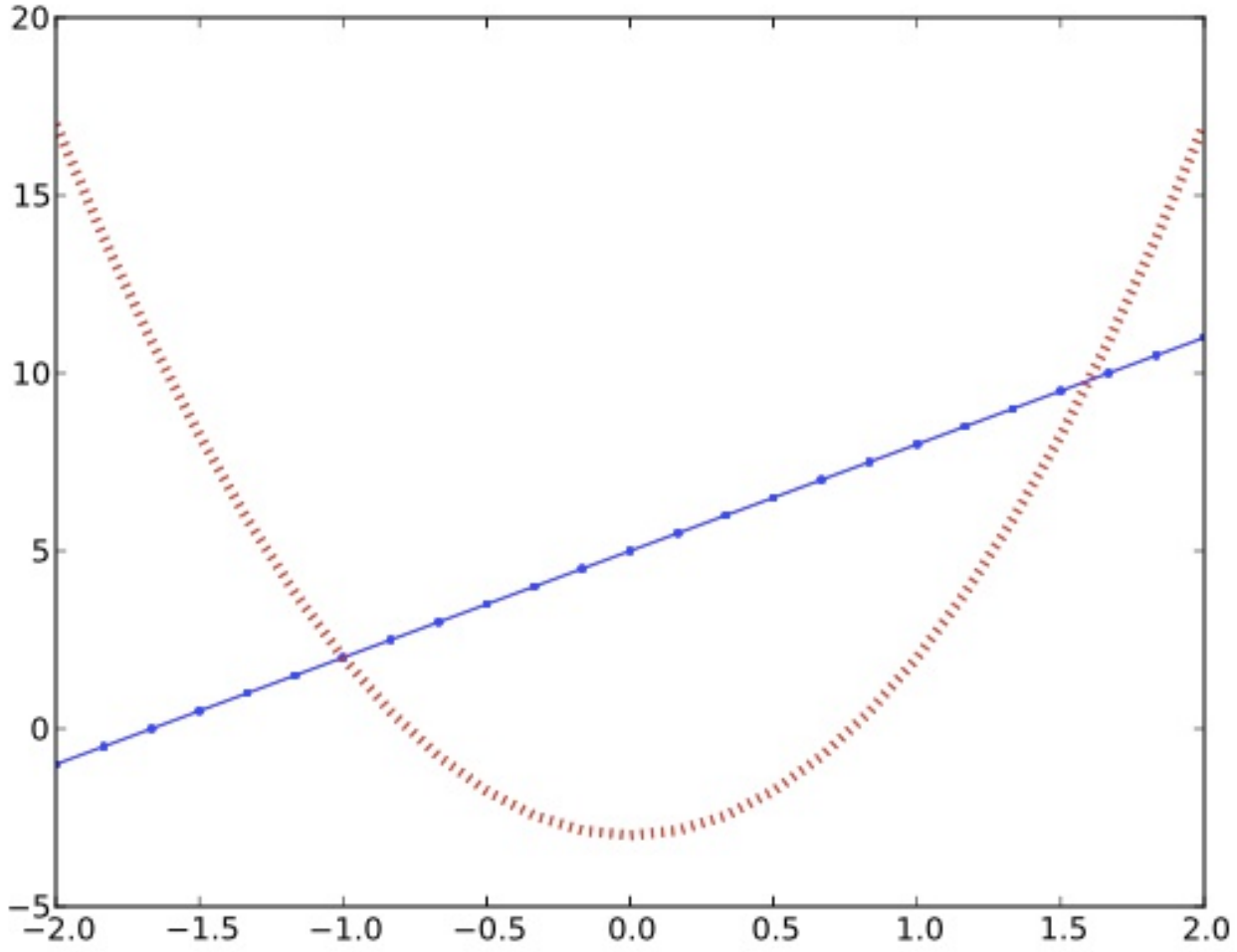
Kleuren en markers

- `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 (verv.)

```
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()

exit(0)
```

Kleuren en markers (verv.)



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

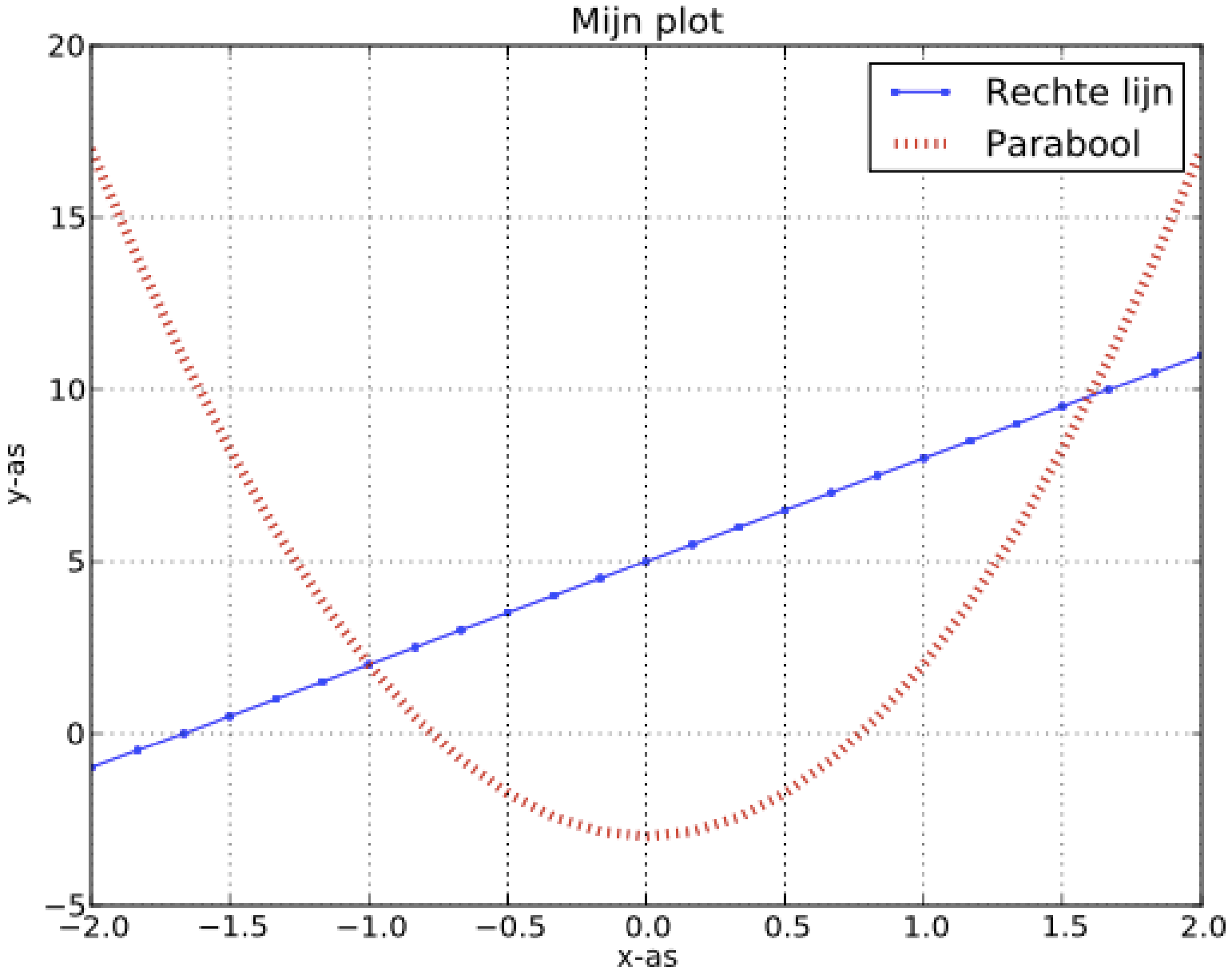
- Met `plt.grid(True)` kun je een achtergrond grid aanzetten.
- De intervallen van de assen kunnen op verschillende manieren worden ingesteld:
  - `plt.ylim(-2, 10)` en analoog voor `plt.xlim()`.
  - Of: `plt.axis(xmin=0, xmax=20., ymin=-10, ymax=100.)`.
- `plt.xscale("log")`: geef de x-as een logaritmische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

exit(0)
```


Mijn plot
Rechte lijn
Parabool
y-as
x-as
20
15
10
5
0
−5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

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.
- Hoe beginnen we nu een nieuw figuur?
- Functie: `plt.figure()`.

Workflow

1) plt.figure().
2) Een of meerdere aanroepen plt.plot().
3) Plot opmaken door assen in te stellen, titel te zetten, enz.
4) plt.show() of plt.savefig().
5) Optioneel: terug naar stap 1 voor de volgende plot.

Eindopracht

- Je weet nu genoeg om de gehele eindopdracht te maken.
- Met de kennis van volgende week kan je een aantal zaken nog wel mooier en handiger programmeren.

Volgende week

- Meer NumPy
  - Drie en meer dimensies
  - Meer array operaties
  - Random numbers
- Scatter plots & histogrammen
- Hoe meer leren over Python?

Morgen werkcollege

- 11:15 - 13:00 uur
- Snelliusgebouw: zalen 303-308
- Aanwezigheidscontrole
- 1 verplichte opdracht voor 0.5 punt (laatste keer)
- Opgaven komen in de loop van vandaag online.
- Werk al aan de eindopdracht! Wacht niet tot volgende week!