Prof. Dr. Gemma Roig
M.Sc. Alperen Kantarcı
M.Sc. Gamze Akyol

Programmieren für Studierende der Naturwissenschaften

Lecture 8 – Handling external data and visualization

Contents

- L6: External Packages, Introduction NumPy and SciPy
  P6: Exercises

- L7:External Packages 2
  P7: Exercises

- L8: Handling external data and visualization
  P8: Exercises

- L9: Design of algorithms
  P9: Exercises (not graded) and independent work in small groups

Towards to the end

- You already know the most important language elements (rest is easy if you want to continue to program)
- We also discussed the basics of working with files
- In other programming languages, it will look different!

BUT:

- Structures (loops, branches, functions)
- Principles(variables, datatypes, allocation, memory)
- ...you will meet again in other languages (maybe with slightly different rules)

Today: More practice with data

- We will not be going through linear regression
- Plotting images
- Python List comprehension (functional programming, and advanced for loop)
- Pandas for data science

Generating larger test quantities - approach to trial and error

- Artificially generate test data, for this:
    - create linearly distributed values (np.linspace)
    - Insert into a linear function (set parameters before hand)
    - "perturb” values (add noise as if they were real values) (random or gaussian)

Coincidence?

numpy.random.randn

numpy.random.randn(*d0*, *d1*, ..., *dn*)

Return a sample (or samples) from the "standard normal" distribution.

If positive, int_like or int-convertible arguments are provided, randn generates an array of shape `(d0, d1, ..., dn)`, filled with random floats sampled from a univariate "normal" (Gaussian) distribution of mean 0 and variance 1 (if any of the d_i are floats, they are first converted to integers by truncation). A single float randomly sampled from the distribution is returned if no argument is provided.

This is a convenience function. If you want an interface that takes a tuple as the first argument, use numpy.random.standard_normal instead.

Parameters: **d0, d1, ..., dn** : *int, optional*
The dimensions of the returned array, should be all positive. If no argument is given a single Python float is returned.

Returns: **Z** : *ndarray or float*
A `(d0, d1, ..., dn)`-shaped array of floating-point samples from the standard normal distribution, or a single such float if no parameters were supplied.

See also:
random.standard_normal Similar, but takes a tuple as its argument.

Notes

For random samples from $N(\mu, \sigma^2)$, use:

```
sigma * np.random.randn(...) + mu
```

Examples

```
>>> np.random.randn()
2.1923875335537315 #random
```

Two-by-four array of samples from N(3, 6.25):

```
>>> 2.5 * np.random.randn(2, 4) + 3
array([[-4.49401501,  4.00950034, -1.81814867,  7.29718677],  #random
       [ 0.39924804,  4.68456316,  4.99394529,  4.84057254]]) #random
```

Another example - images

- How can I work with images in Python?

- Things you might want to do with pictures:
  - Output them (look at them)
  - Look at (and maybe even change) the histogram (brightness distribution)?
  - Add filter to your photos
  - Calculate the Fourier transform (frequencies)?

- How do you get started if you don’t know anything about it?
  - Search and try the simplest examples at the beginning
  - Modify examples and make them more complicated
  - Set and solve your own tasks from practice

What is an image?

- Just too many numbers (640x480 = 3 millions)
- They are organized in a structured way (spatiality)

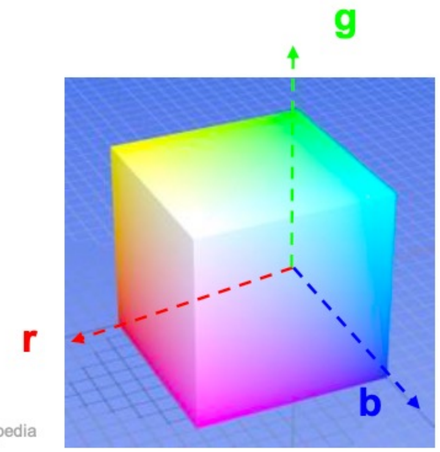
How to represent color in computer?

In standard colour photographic images, one uses 8 bits for *each* colour channel (red, green, blue), or 24 bits per pixel. That means there are 2^{24} possible colours for a pixel. This is roughly 16.7 million colours!

How to represent color in computer?

RGB colour space

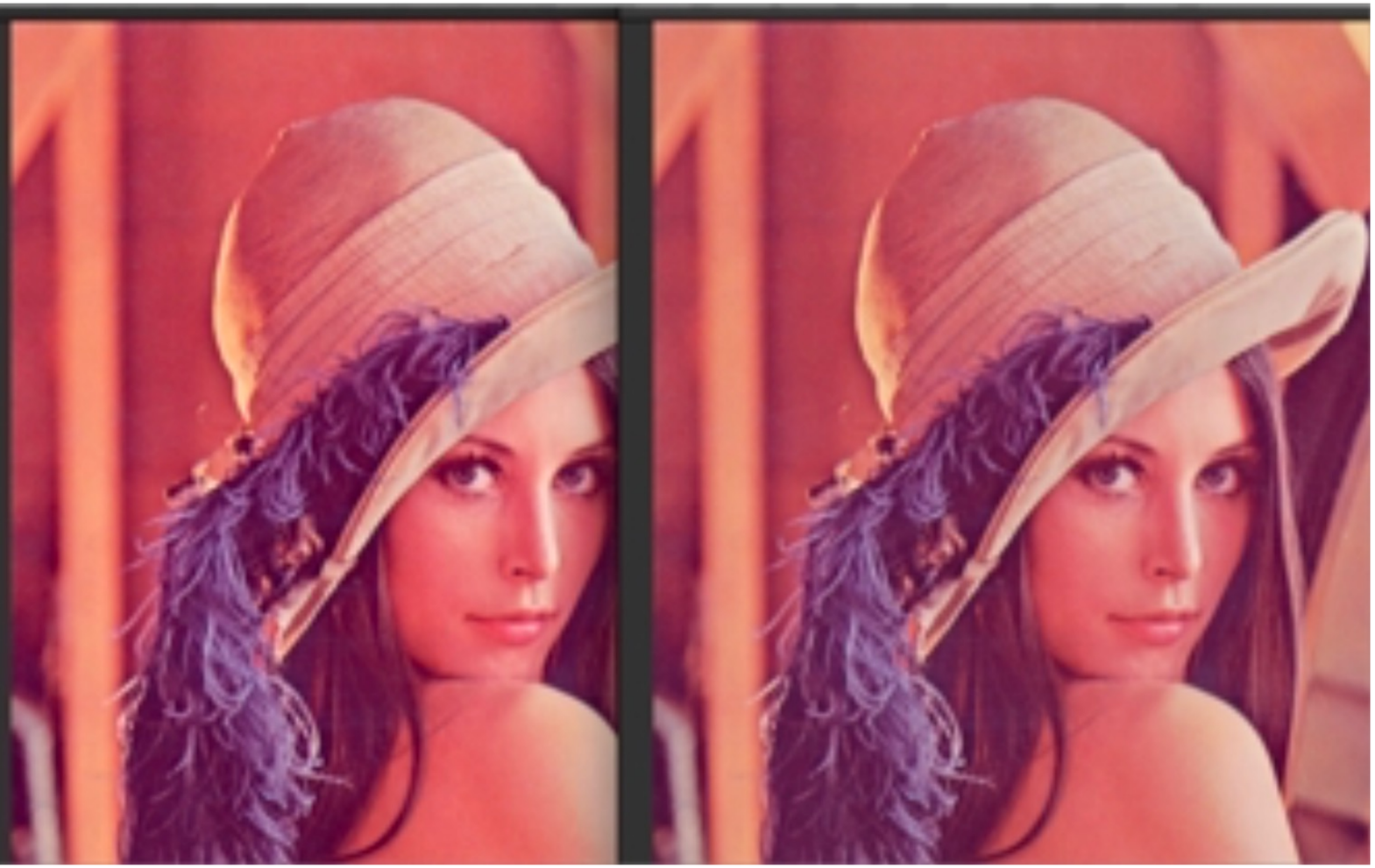
- Has its origin in colour television, now used in displays (flat panels, phones)
- *Additive* mixing or red, green, and blue light form the final colour
- RGB is a colour space
  - Red axis, green axis, blue axis
  - Values typically in the range from 0 (none) to 255 (full colour) along each axis
  - A colour is a point in this space, represented as a vector $[r, g, b]^T$



Source: wikipedia

Example Image Manipulation

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on

@author: alexanderwolodkin
"""

from PIL import Image
from PIL import ImageEnhance



img = Image.open("lena.png")
img.show()

enhancer = ImageEnhance.Contrast(im
enhancer.enhance(0.9).show()
```

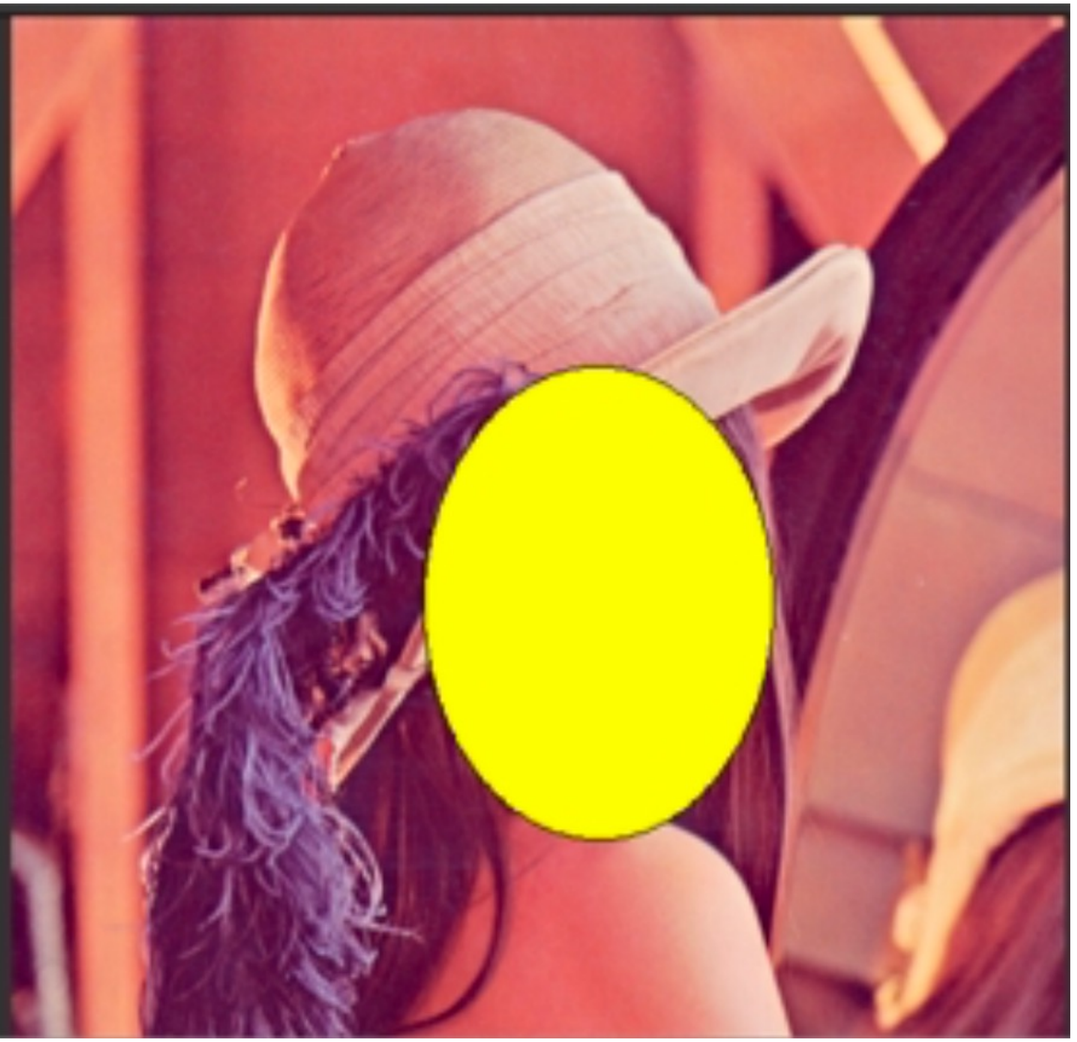


Example Image Manipulation

```
# -*- coding: utf-8 -*-
"""
Created on

@author: alexanderwolodkin
"""

from PIL import Image
img = Image.open("lena.png")
# print()
# print(img)
# r, g, b = img.split()
# print(r)

#testR = Image.merge("RGB",
 #                   (g,b,r))

img = img.rotate(45, expand=True)
img.show()

```

Example Image Manipulation

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on

@author: alexanderwolodkin
"""

from PIL import Image
# from PIL import ImageEnhance
from PIL import ImageDraw

img = Image.open("lena.png")
# img.show()

test = ImageDraw.Draw(img)
test.ellipse((200,170,370,400),
             'yellow','black')

img.show()
```



Example Image Manipulation

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on

@author: alexanderwolodkin
"""

from PIL import Image
# from PIL import ImageEnhance
from PIL import ImageDraw

img = Image.open("lena.png")

width, height = img.size
for x in range(int(width/2)):
    for y in range(height):
        r, g, b = img.getpixel((x,
        r, g, b = r, int(g*1.5), b
        img.putpixel((x,y), (r, g,

for x in range(int(width/2), width)
    for y in range(height):
        r, g, b = img.getpixel((x,
        r, g, b = r, g, int(b*1.5)
        img.putpixel((x,y), (r, g,
img.show()
```

```
ase -- a string containing
CII uppercase letters
containing all ASCII deci
octdigits -- a string
all ASCII punctuation
idered printable

thon/temp.py', wdir=
```

Example Image Manipulation

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on

@author: alexanderwolodkin
"""

from PIL import Image
# from PIL import ImageEnhance
from PIL import ImageDraw

img = Image.open("lena.png")

width, height = img.size
for x in range(width):
    for y in range(height):
        r, g, b = img.getpixel((x,
        grey = int((r + g + b) / 3)
        img.putpixel((x,y), (grey,
img.show()
```

```
ase -- a string containing
SCII uppercase letters
g containing all ASCII deci
octdigits -- a string
all ASCII punctuation
sidered printable

base

/thon/temp.py', wdir=
Users/alexanderwolodkin/Documents/Python')
```

Python List Comprehension

- List comprehension lets you create a list while using for loops, if/else and functions.
- It offers a shorter syntax when you want to create a new list based on the values of an existing list.

The syntax is:

```
newlist = [expression for item in iterable if condition == True]
```

```
newlist = [x for x in range(10)]
newlist = [x for x in range(10) if x < 5]
```

Python List Comprehension

```
num_list = [y for y in range(100) if y % 2 == 0 if y % 5 == 0]
print(num_list)
```

or you can do the same with

```
num_list = [y for y in range(100) if y % 2 == 0 and y % 5 == 0]
print(num_list)
```

You can also iterate over different lists

```
list1 = range(100)
list2 = [i*20 for i in list1]
print(list2)
```

```
[0, 20, 40, 60, 80, 100, 120, 140, 160, 180, 200, 220, 240, 260, 280, 300, 320, 340, 360, 380, 400, 420, 440, 4
60, 480, 500, 520, 540, 560, 580, 600, 620, 640, 660, 680, 700, 720, 740, 760, 780, 800, 820, 840, 860, 880, 90
0, 920, 940, 960, 980, 1000, 1020, 1040, 1060, 1080, 1100, 1120, 1140, 1160, 1180, 1200, 1220, 1240, 1260, 1280
, 1300, 1320, 1340, 1360, 1380, 1400, 1420, 1440, 1460, 1480, 1500, 1520, 1540, 1560, 1580, 1600, 1620, 1640, 1
660, 1680, 1700, 1720, 1740, 1760, 1780, 1800, 1820, 1840, 1860, 1880, 1900, 1920, 1940, 1960, 1980]
```

Python List Comprehension

```
fruits = ["apple","grape","orange","carrot","strawberry"]
capital_fruits = []
print(fruits, capital_fruits)
for fruit in fruits:
    capital_fruits.append(fruit.upper())

print(fruits,capital_fruits)
```

```
========== RESTART: C:/Users/alperen/Desktop/list_comprehension.py ==========
['apple', 'grape', 'orange', 'carrot', 'strawberry'] []
['apple', 'grape', 'orange', 'carrot', 'strawberry'] ['APPLE', 'GRAPE', 'ORANGE', 'CARROT', 'STRAWBERRY']
```

What we do is create a list and populate it with elements.

We can do the same operation more easily with list comprehension.

Python List Comprehension

```
fruits = ["apple","grape","orange","carrot","strawberry"]
capital_fruits = []
print(fruits, capital_fruits)
for fruit in fruits:
    capital_fruits.append(fruit.upper())

print(fruits,capital_fruits)



capital_fruits_comprehension = [ fruit.upper() for fruit in fruits]


print(fruits,capital_fruits_comprehension)
```

Pandas

- We have seen that we have different data formats as inputs (txt, csv, FASTA, image, …)
- If you are dealing with tabular data (kind of data you keep in excels), csv files are very common.
- However, dealing with strings, rows, columns is hard (your last exercise) and time consuming. You have to manipulate the strings, keep them in different lists, loop over them etc.
- Most of the time we apply the same operation on all rows, or calculate the same data statistics on all rows.
- You import pandas (and numpy) as we did like before:

```
import numpy as np
import pandas as pd
```

Pandas data types

As we know, each module provides you some custom data structure (data type).

Pandas provides Series and DataFrame.

```
s = pd.Series([1, 3, 5, np.nan, 6, 8])
s
0    1.0
1    3.0
2    5.0
3    NaN
4    6.0
5    8.0
dtype: float64
```

Pandas data types

You can think Series as a one column, and DataFrame as a whole Excell table.
Each row has either name or index, each column has either name or index.

```
dates = pd.date_range("20230101", periods=6)
print(dates)
DatetimeIndex(['2023-01-01', '2023-01-02', '2023-01-03', '2023-01-04',
               '2023-01-05', '2023-01-06'],
              dtype='datetime64[ns]', freq='D')
df = pd.DataFrame(np.random.randn(6, 4), index=dates, columns=list("ABCD"))
print(df)
                   A         B         C         D
2023-01-01  0.257303  0.403990  0.702875 -0.331377
2023-01-02 -0.534463 -0.469239 -1.691661  2.005961
2023-01-03  0.186946 -0.541249 -0.801999  1.724059
2023-01-04  0.319749 -0.215950 -0.256406 -1.100276
2023-01-05  0.337620 -0.914500 -1.559577 -0.058207
2023-01-06  0.597004 -1.536332 -0.786414 -0.460383
```

Reading csv with Pandas

```
df = pd.read_csv("C:\\Users\\alperen\\Desktop\\Namen.csv")
print(df)
```

	ID	Vorame	Nachname	Alter
0	1	Max	Mueller	27
1	2	Lisa	Muster	19
2	3	Severin	Klein	11
3	4	Harry	Klas	44
4	5	Hermine	Jakob	34
5	6	Ron	Riedel	52
6	7	Andreas	Rustig	6
7	8	Andreas	Hagen	42
8	9	Fromut	Haagen	24
9	10	Merle	Klaus	52
10	11	Meike	Klaus	45
11	12	Maike	Otto	24
12	13	Meike	Wert	18
13	14	Lissi	Weil	89
14	15	Barbara	Dursley	32
15	16	Helena	Agabe	54
16	17	Ole	Viel	23
17	18	Ole	Viehl	67
18	19	Ernst	Klammer	98
19	20	Emil	Klast	32
20	21	Emilia	Franz	18
21	22	Rosi	Fried	13
22	23	Susanne	Frost	92
23	24	Hannah	Meyer	42
24	25	Hannah	Schmitt	5
25	26	Anja	Schmidt	45
26	27	Adalbert	Weilbrunn	12
27	28	Angmar	Gutenberg	41
28	29	Frodo	Greve	65

Ac

Reading csv with Pandas

```
df.head()
```

	ID	Vorame	Nachname	Alter
0	1	Max	Mueller	27
1	2	Lisa	Muster	19
2	3	Severin	Klein	11
3	4	Harry	Klas	44
4	5	Hermine	Jakob	34

```
df.head()
```

	ID	Vorame	Nachname	Alter
0	1	Max	Mueller	27
1	2	Lisa	Muster	19
2	3	Severin	Klein	11
3	4	Harry	Klas	44
4	5	Hermine	Jakob	34

Reading csv with Pandas

```
> df.iloc[3:5, 0:2]
   ID   Vorame
3   4    Harry
4   5  Hermine
```

Reading csv with Pandas

```
df["Nachname"]
0       Mueller
1        Muster
2         Klein
3          Klas
4         Jakob
5        Riedel
6        Rustig
7         Hagen
8        Haagen
9         Klaus
10        Klaus
11         Otto
12         Wert
13         Weil
14      Dursley
15        Agabe
16         Viel
17        Viehl
18      Klammer
19        Klast
20        Franz
21        Fried
22        Frost
23        Meyer
24      Schmitt
25      Schmidt
26    Weilbrunn
27    Gutenberg
28        Greve
Name: Nachname, dtype: object
```

Summary

- Trying new things can be tedious:

- Problem → Lookup
- Minimal examples → errors → correction → repeat forever...
- Reading documentation is important to understand the syntax and function of individual commands!
- Start with small examples and try to abstract from there!