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

Programmieren für Studierende der Naturwissenschaften

Lecture 7 – Reading files and external packages 2

Contents

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

- L7: Reading files and 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

Questions from last lectures and how to practice

In case the content is too fast....

- Better to repeat tasks from last week again.
- Feel free to ask me and discuss with other participants
- In particular, the use of loops and conditions should work by now.
- Find the easy tasks from the exercises.

- Use online resources to practice your skills
- Content wise not too much more will be done in the next lecture (therefore a bit shorter). Use the time to practice!

Self-practice

If you want to make more practice (you should if you want to keep the knowledge),
You can do the following:

- Think of a problem in your mind and try to code it
- Think of small games, or programs that might be useful. Implement them.
- Write a basic calculator program in Python that can compute basic arithmetics, stores values in memory, etc. (You can do that with your current knowledge)
- Force yourself to use different modules. PyGame, BioPython, Custom Modules from your area of expertise

- If you cannot think of something to code:
  - We have some online resources that give you infinite number of problems to solve.
  - Learning and solving algorithm problems is a **different task** than learning programming
  - Hackerrank (highly suggested), codingame (a bit more advanced), leetcode (mostly algorithms, you can be a software engineer after solving all of the website)

Working with files

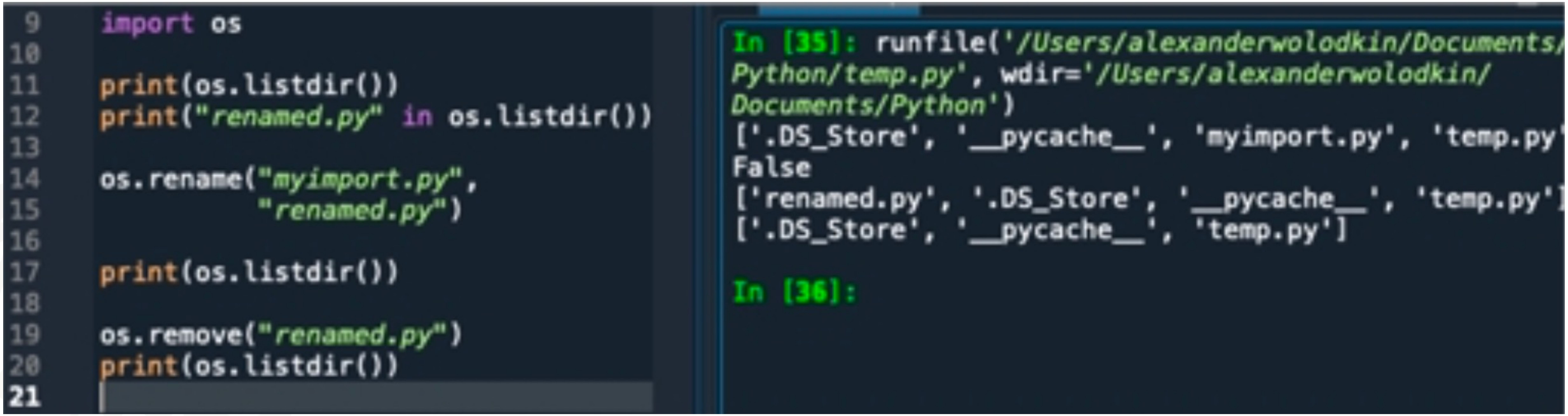
- Datasets that you work with within the natural sciences are often in files
- So, for the analysis and processing of this data, you have to work with files (copying and pasting text is too cumbersome for large data sets)
- Data is fortunately often text-based (e.g..txt, .csv, .html or special/subject-specific formats such as FASTA in biology)
- But also other formats are sometimes available,e.g., these so-called binaries (somewhat more difficult to edit) binary code is a machine language code
- Python already provides modules that simplify reading and saving files

os module saves you from the agony

- The os module offers many functionalities, e.g.
  - `os.rename()` to rename files or folders
  - `os.mkdir()` to create folders
  - `os.path.exists()` to check the existence of a path
  - `os.remove()` to delete files
  - `os.listdir()` to output a list of files and folders

  - etc.

os module saves you from the agony



```python
import os

print(os.listdir())
print("renamed.py" in os.listdir())

os.rename("myimport.py",
          "renamed.py")

print(os.listdir())

os.remove("renamed.py")
print(os.listdir())
```

```
In [35]: runfile('/Users/alexanderwolodkin/Documents/
Python/temp.py', wdir='/Users/alexanderwolodkin/
Documents/Python')
['.DS_Store', '__pycache__', 'myimport.py', 'temp.py'
False
['renamed.py', '.DS_Store', '__pycache__', 'temp.py']
['.DS_Store', '__pycache__', 'temp.py']

In [36]:
```

Open text files

- With the command `open(<filename>.<ending>,[<typetoopen>])`

- There are options for access (e.g. reading('r’), writing('w’), binaries('b’), etc.).

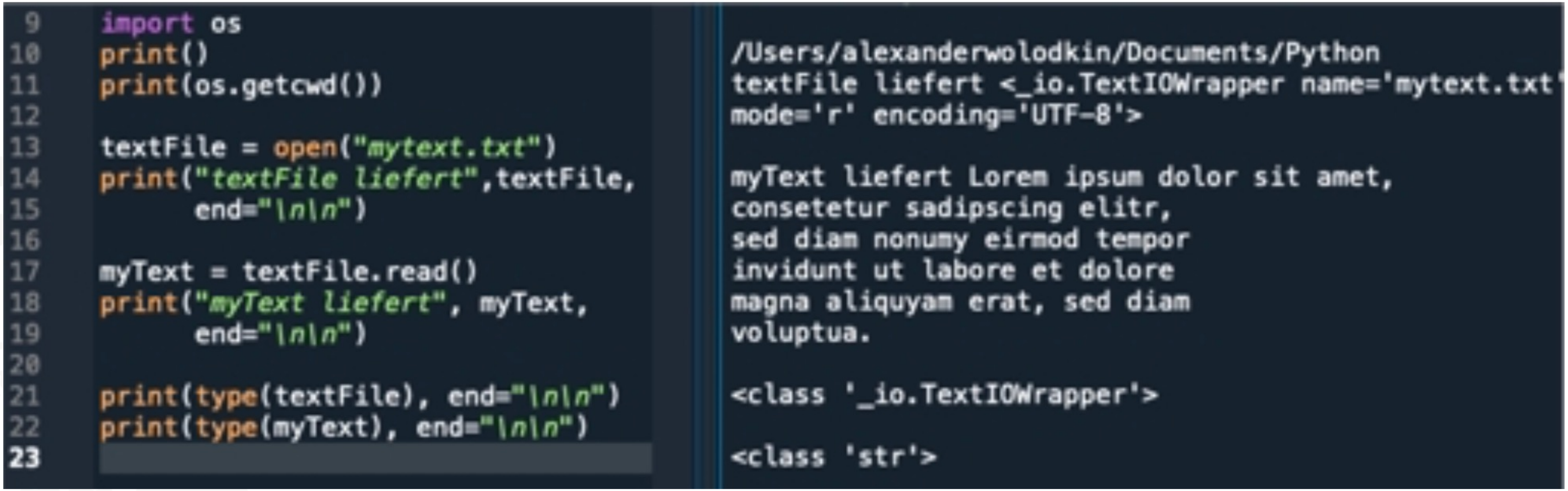
- Look it up on your own!

- If the file is not in the current path, the path must be specified before the filename

- The specified file will be opened only-creates a file-object

- With`<variablename>.read()`the content is read and can be stored as a string in a variable

Example



```python
import os
print()
print(os.getcwd())

textFile = open("mytext.txt")
print("textFile liefert",textFile,
      end="\n\n")

myText = textFile.read()
print("myText liefert", myText,
      end="\n\n")

print(type(textFile), end="\n\n")
print(type(myText), end="\n\n")
```

```
/Users/alexanderwolodkin/Documents/Python
textFile liefert <_io.TextIOWrapper name='mytext.txt'
mode='r' encoding='UTF-8'>

myText liefert Lorem ipsum dolor sit amet,
consetetur sadipscing elitr,
sed diam nonumy eirmod tempor
invidunt ut labore et dolore
magna aliquyam erat, sed diam
voluptua.

<class '_io.TextIOWrapper'>

<class 'str'>
```

Separators in textfiles

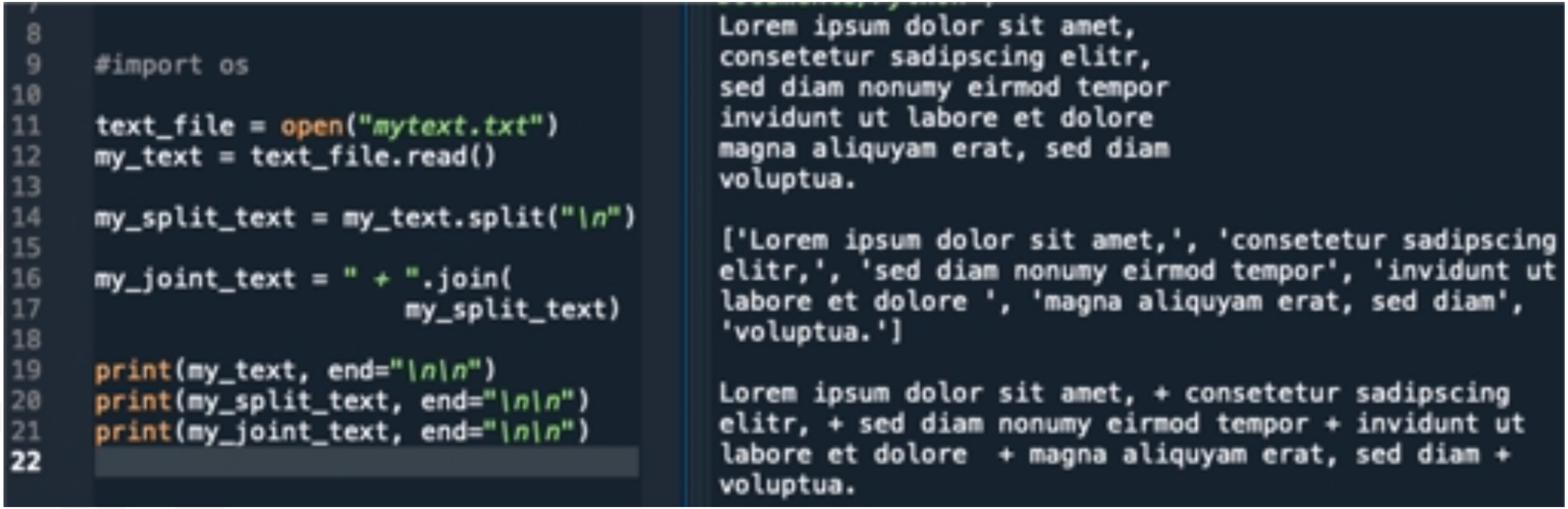
- Many text files contain "separators” to separate ”table entries” from each other, e.g. .csv(comma- separated-value)

- However, this depends on the dataset

- First of all, familiarize yourself with the dataset and its contents

- With common string operations you can then work on the contents of text files
  - e.g.slicing, split, join, removing whitespaces etc.

Example



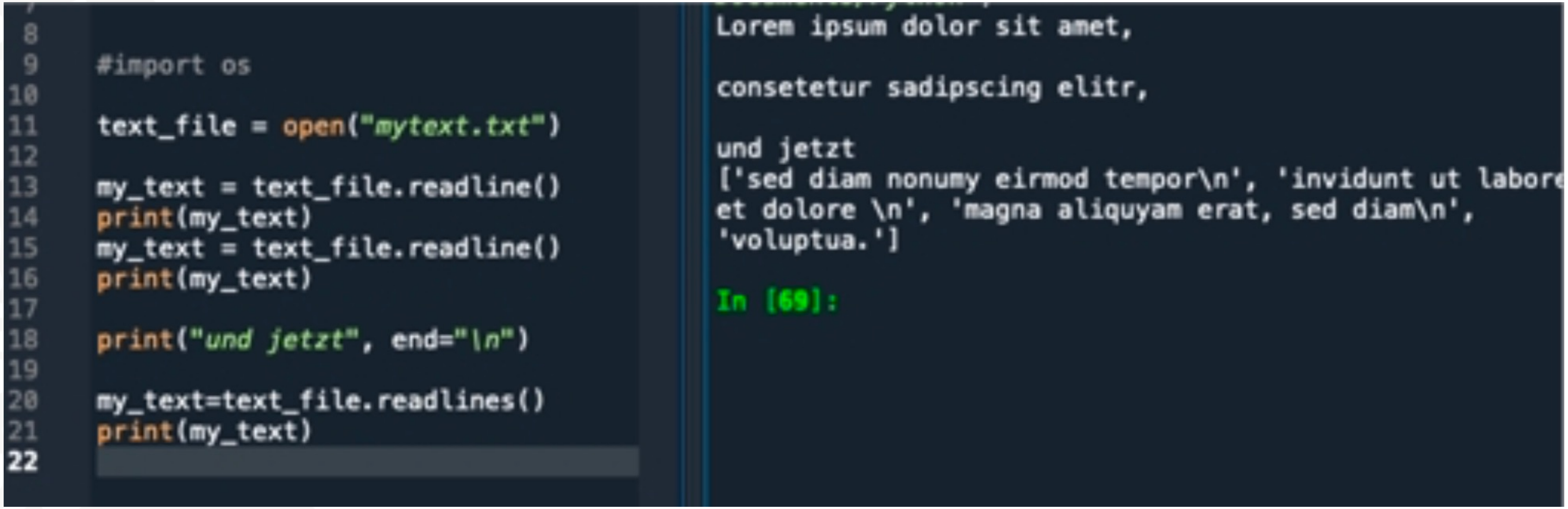
```
#import os

text_file = open("mytext.txt")
my_text = text_file.read()

my_split_text = my_text.split("\n")

my_joint_text = " + ".join(
                  my_split_text)

print(my_text, end="\n\n")
print(my_split_text, end="\n\n")
print(my_joint_text, end="\n\n")
```

```
Lorem ipsum dolor sit amet,
consetetur sadipscing elitr,
sed diam nonumy eirmod tempor
invidunt ut labore et dolore
magna aliquyam erat, sed diam
voluptua.

['Lorem ipsum dolor sit amet,', 'consetetur sadipscing
elitr,', 'sed diam nonumy eirmod tempor', 'invidunt ut
labore et dolore ', 'magna aliquyam erat, sed diam',
'voluptua.']

Lorem ipsum dolor sit amet, + consetetur sadipscing
elitr, + sed diam nonumy eirmod tempor + invidunt ut
labore et dolore  + magna aliquyam erat, sed diam +
voluptua.
```

Alternative ways to read files

Besides `<name>.read()` there are 2 more possibilities

- `<name>.readline()` reads the first unread line of the file as a string
- `<name>.readlines()` stores each (still unread) line as a string in a list



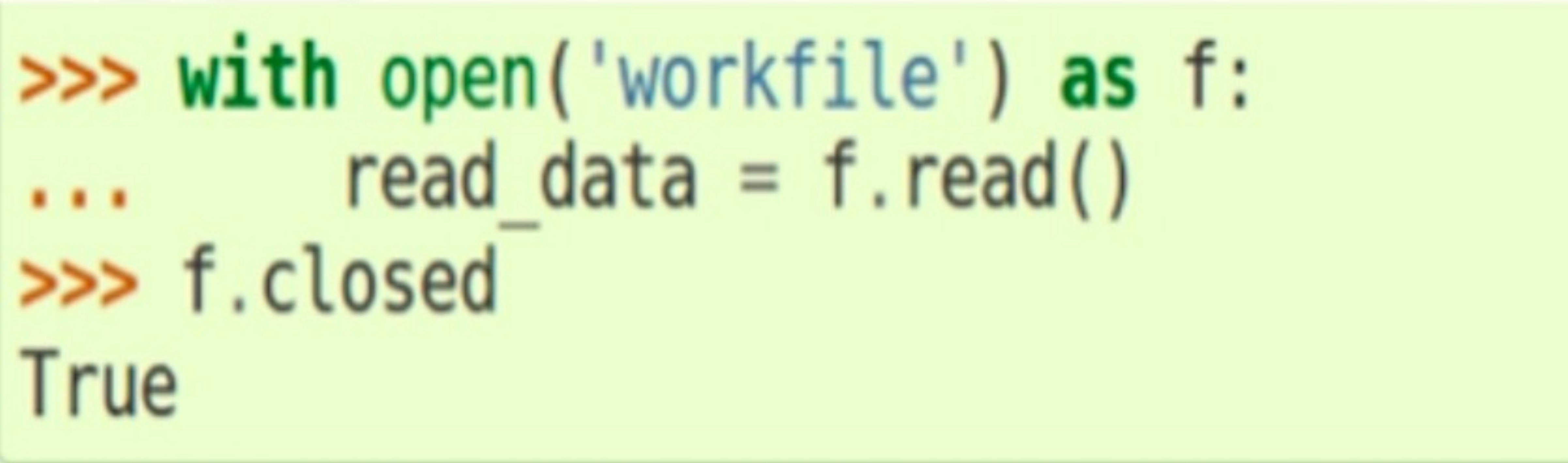
```
#import os

text_file = open("mytext.txt")

my_text = text_file.readline()
print(my_text)
my_text = text_file.readline()
print(my_text)

print("und jetzt", end="\n")

my_text=text_file.readlines()
print(my_text)
```

```
Lorem ipsum dolor sit amet,

consetetur sadipscing elitr,

und jetzt
['sed diam nonumy eirmod tempor\n', 'invidunt ut labore
et dolore \n', 'magna aliquyam erat, sed diam\n',
'voluptua.']

In [69]:
```

Closing the files

- If you have opened a file with `open()` ,it must be closed again after working (otherwise it may consume a lot of memory!).
- This can be done just as easily with the command `<variablename>.close()`

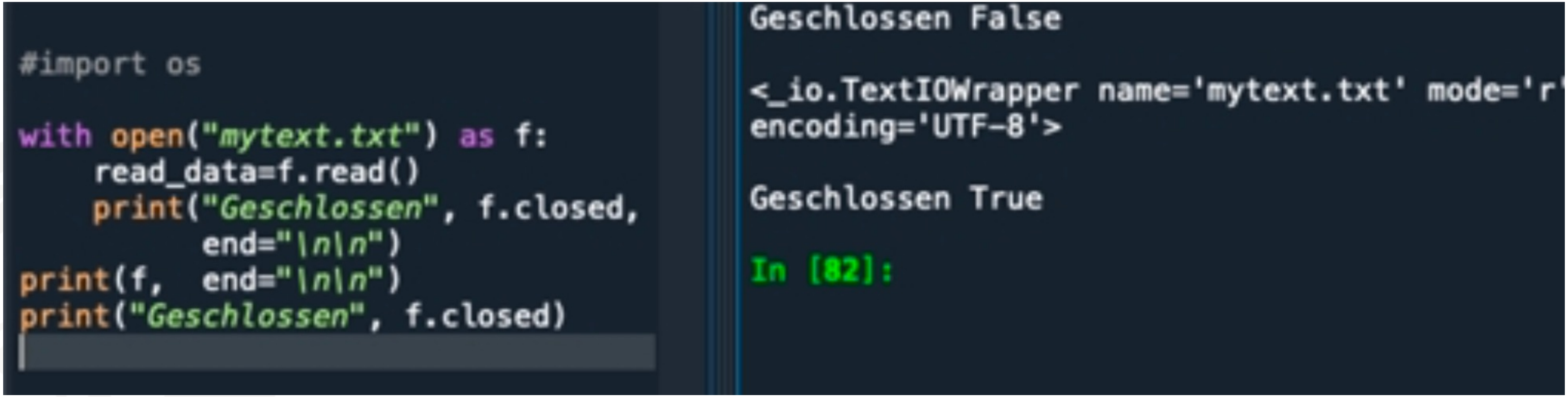
- Alternatively: Python recommends



```
>>> with open('workfile') as f:
...     read_data = f.read()
>>> f.closed
True
```

- The keyword '`with`' closes the file directly after the block has been executed, even if an error has occurred. So you don't forget to close the file.

Example



```
#import os

with open("mytext.txt") as f:
    read_data=f.read()
    print("Geschlossen", f.closed,
          end="\n\n")
print(f,  end="\n\n")
print("Geschlossen", f.closed)
```

```
Geschlossen False

<_io.TextIOWrapper name='mytext.txt' mode='r'
encoding='UTF-8'>

Geschlossen True

In [82]:
```

Writing to the file

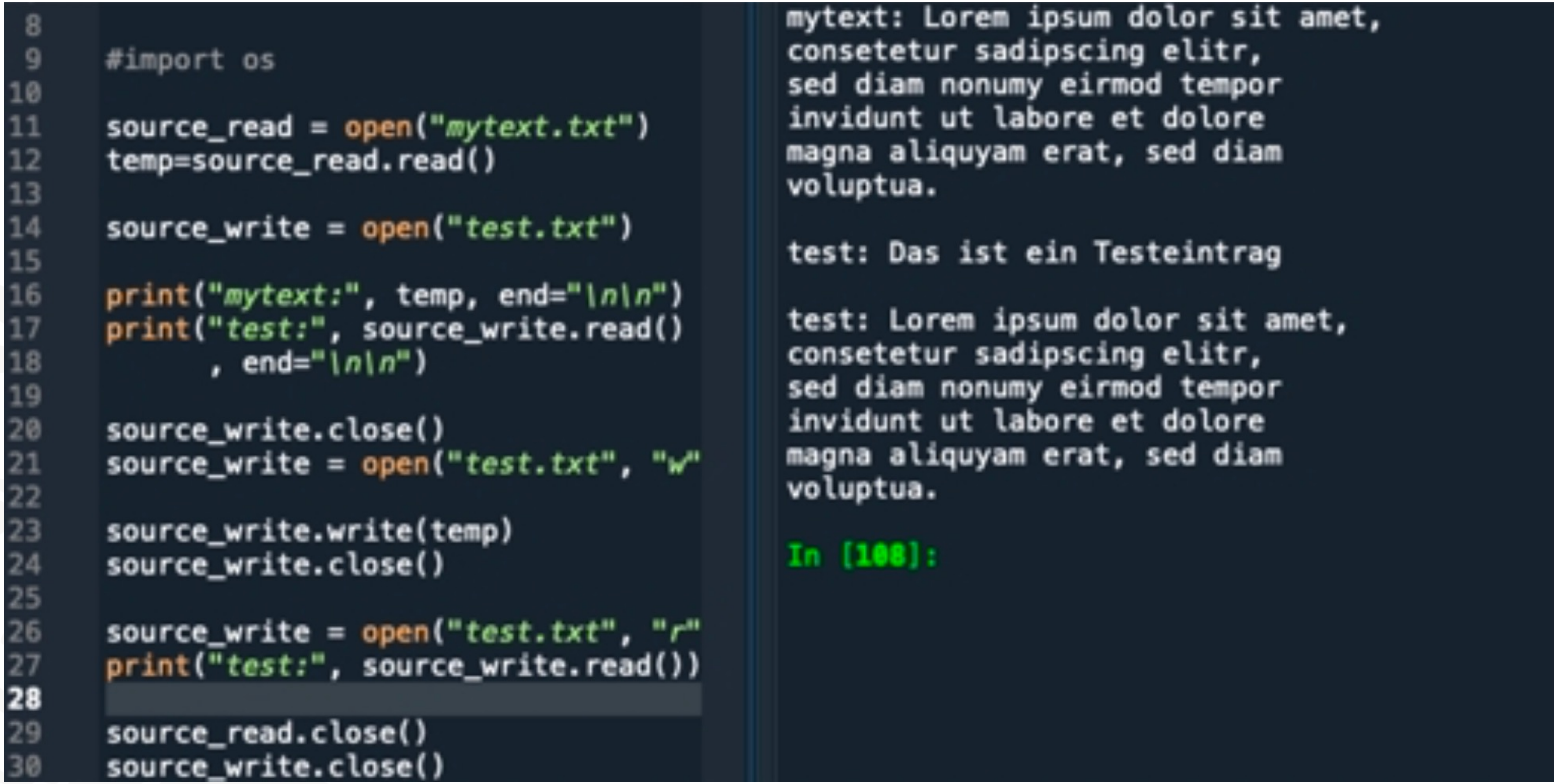
- The `<name>.write(<content>)` function is used to write to the files (and overwrite existing text if necessary).

- For this, the file must have been explicitly opened in write mode.

- So you cannot accidentally overwrite a file

- Info about input and output in the Python doc:
  - https://docs.python.org/3/tutorial/inputoutput.html

Example



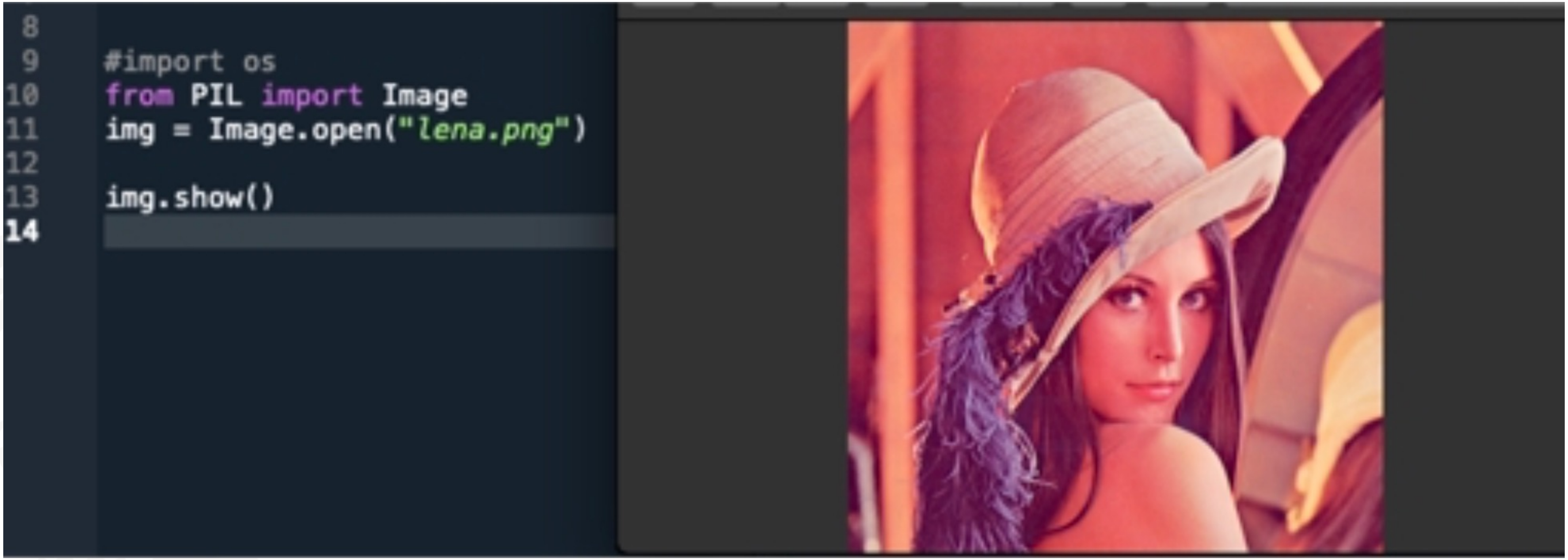
```python

#import os

source_read = open("mytext.txt")
temp=source_read.read()

source_write = open("test.txt")

print("mytext:", temp, end="\n\n")
print("test:", source_write.read()
      , end="\n\n")

source_write.close()
source_write = open("test.txt", "w"

source_write.write(temp)
source_write.close()

source_write = open("test.txt", "r"
print("test:", source_write.read())

source_read.close()
source_write.close()
```

```
mytext: Lorem ipsum dolor sit amet,
consetetur sadipscing elitr,
sed diam nonumy eirmod tempor
invidunt ut labore et dolore
magna aliquyam erat, sed diam
voluptua.

test: Das ist ein Testeintrag

test: Lorem ipsum dolor sit amet,
consetetur sadipscing elitr,
sed diam nonumy eirmod tempor
invidunt ut labore et dolore
magna aliquyam erat, sed diam
voluptua.

In [108]:
```

Other formats than text

- PIL-Python Imaging Library:
- http://www.pythonware.com/library/pil/handbook
- Load, crop, rotate, modify color channels and much more!!!

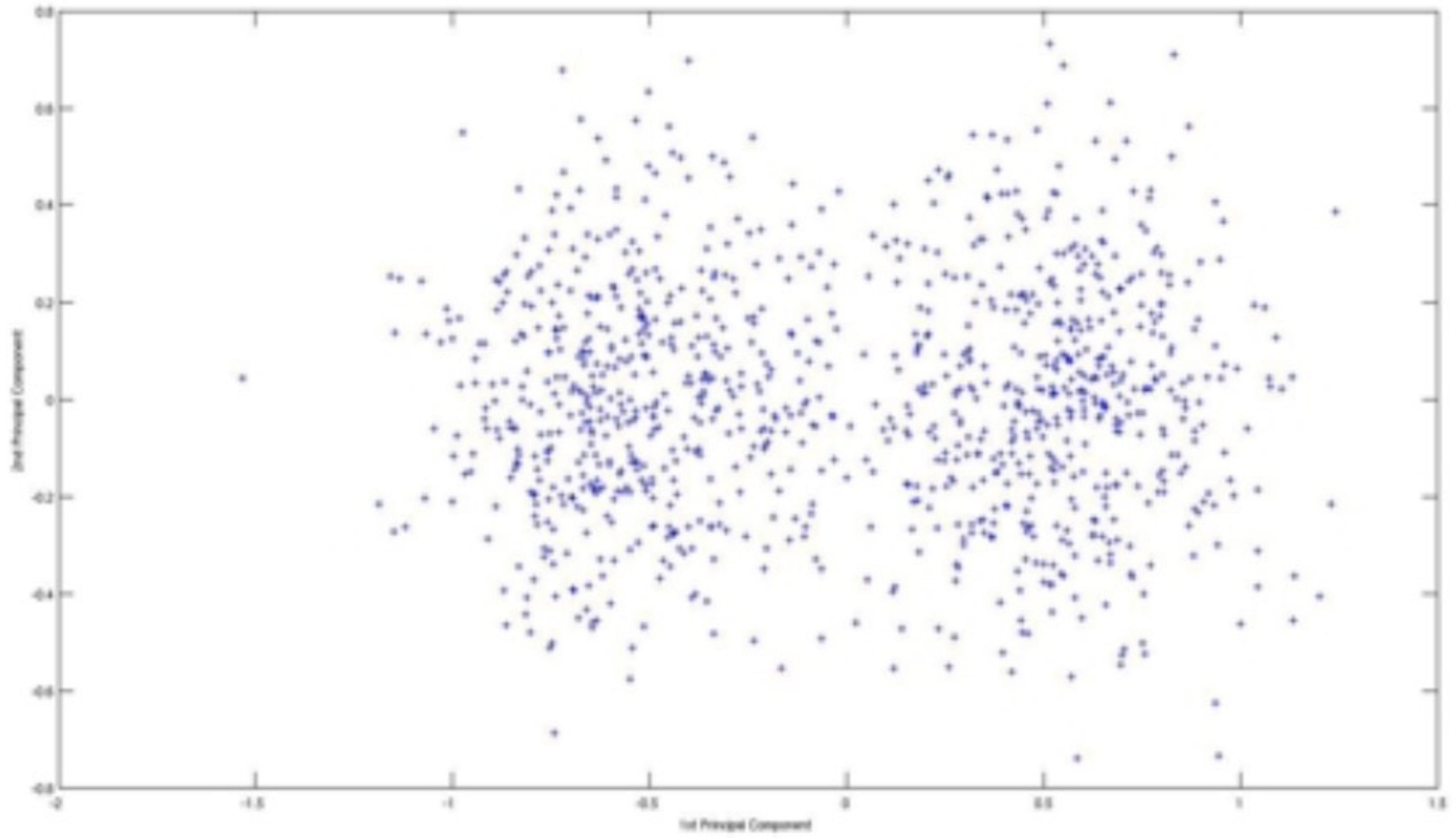
Opening Lena image

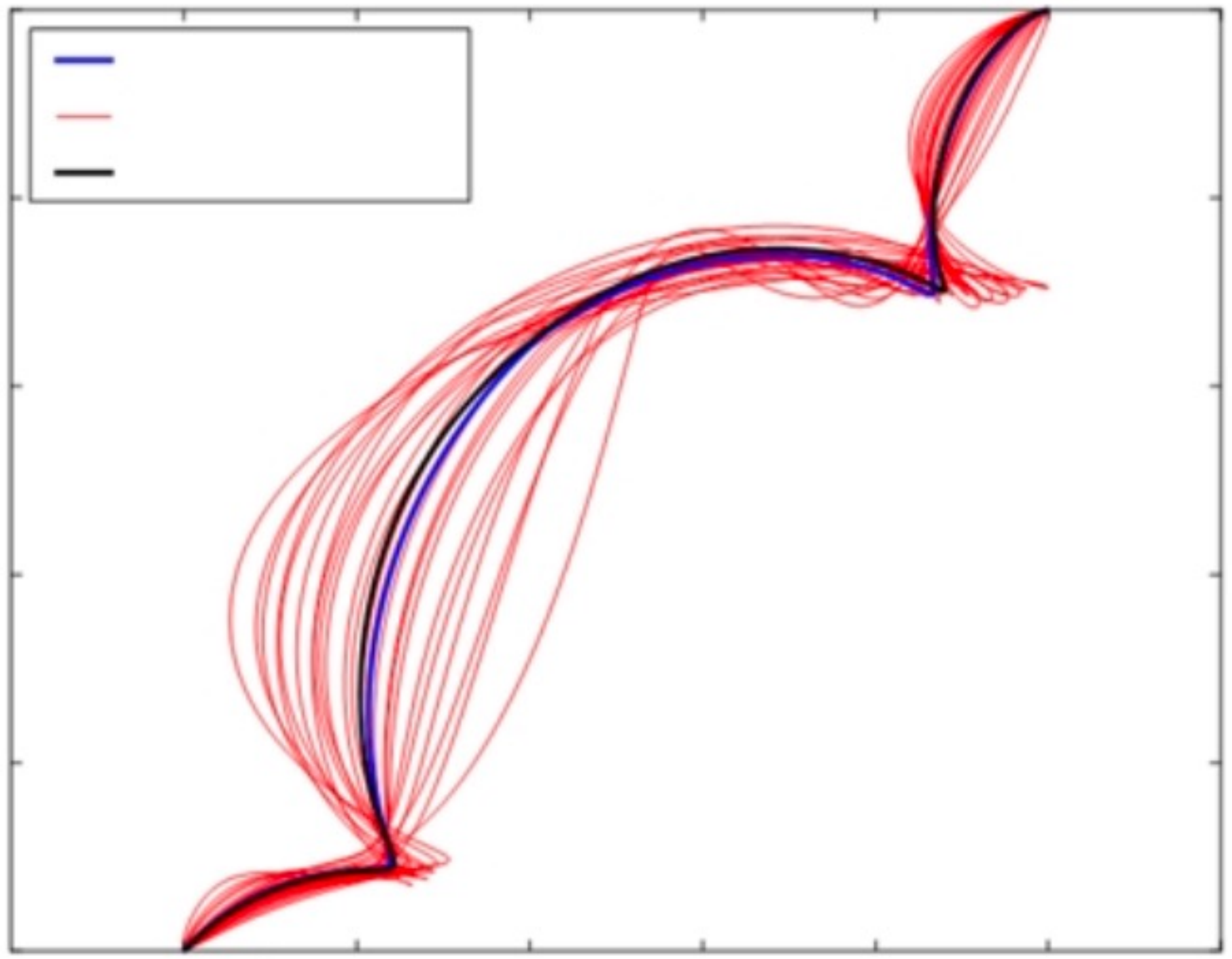


```python

#import os
from PIL import Image
img = Image.open("lena.png")

img.show()

```

Plots with matplotlib

- Often we want to have a graphical representation of data (e.g. to check that our data is in the right frame, to see what history certain values have, or if certain values cluster somewhere)





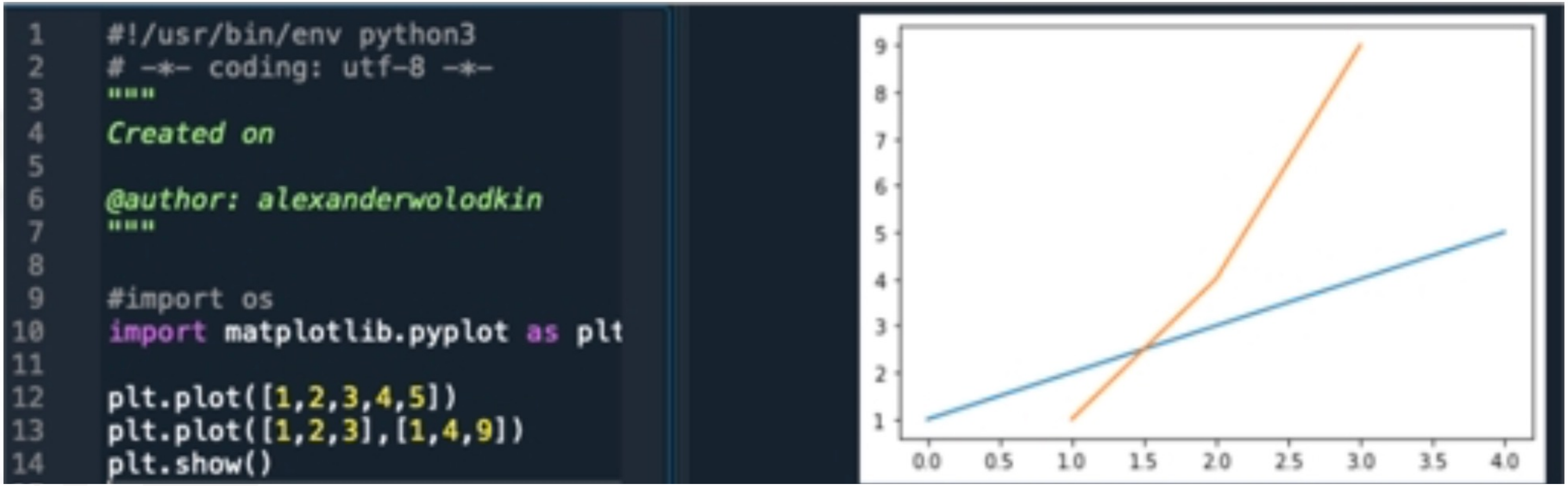
Basics of matplotlib (external module)

- Must be imported. Common instruction is
  - `import matplotlib.pyplot as plt`

- The `plot()` function plots (draws/visualizes) data (specified in the parentheses, possibly other options) but doesn't display it on monitor and then displays it with `plt.show()`

- Simplest example:
  - `plt.plot([1,2,3,4,5])`
  - `plt.plot([1,2,3],[1,4,9])`

Plot example



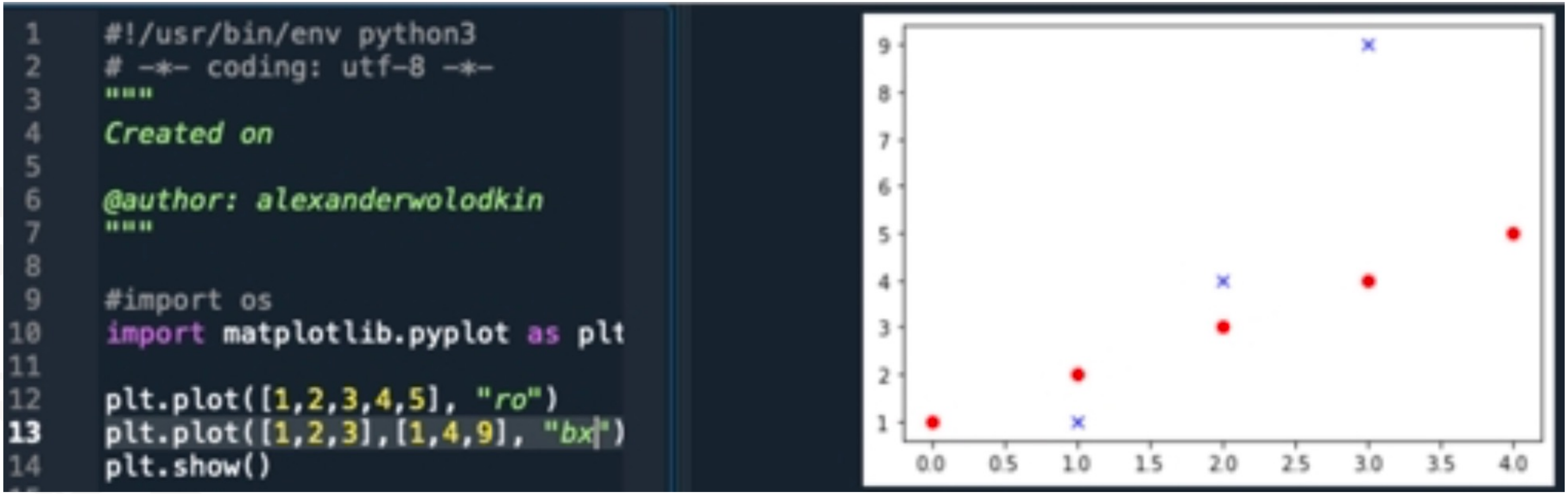
```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on

@author: alexanderwolodkin
"""

#import os
import matplotlib.pyplot as plt

plt.plot([1,2,3,4,5])
plt.plot([1,2,3],[1,4,9])
plt.show()
```

Basics of matplotlib

- Additional options: Color and shape.
- Default: 'b' (blue line), passed as additional argument to the plot() function.
- Othercolors:'c','r'...
- Otherforms:'o','x'...

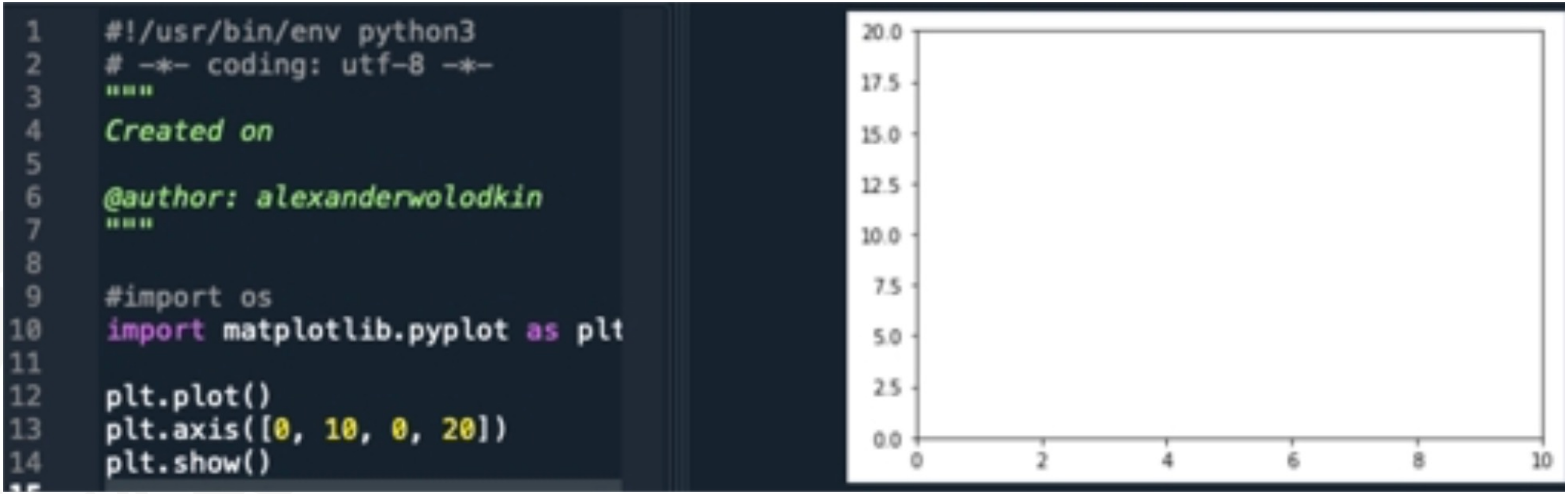
- Additional option: axislength. Is set separately with plt.axis([...])

Basics of matplotlib



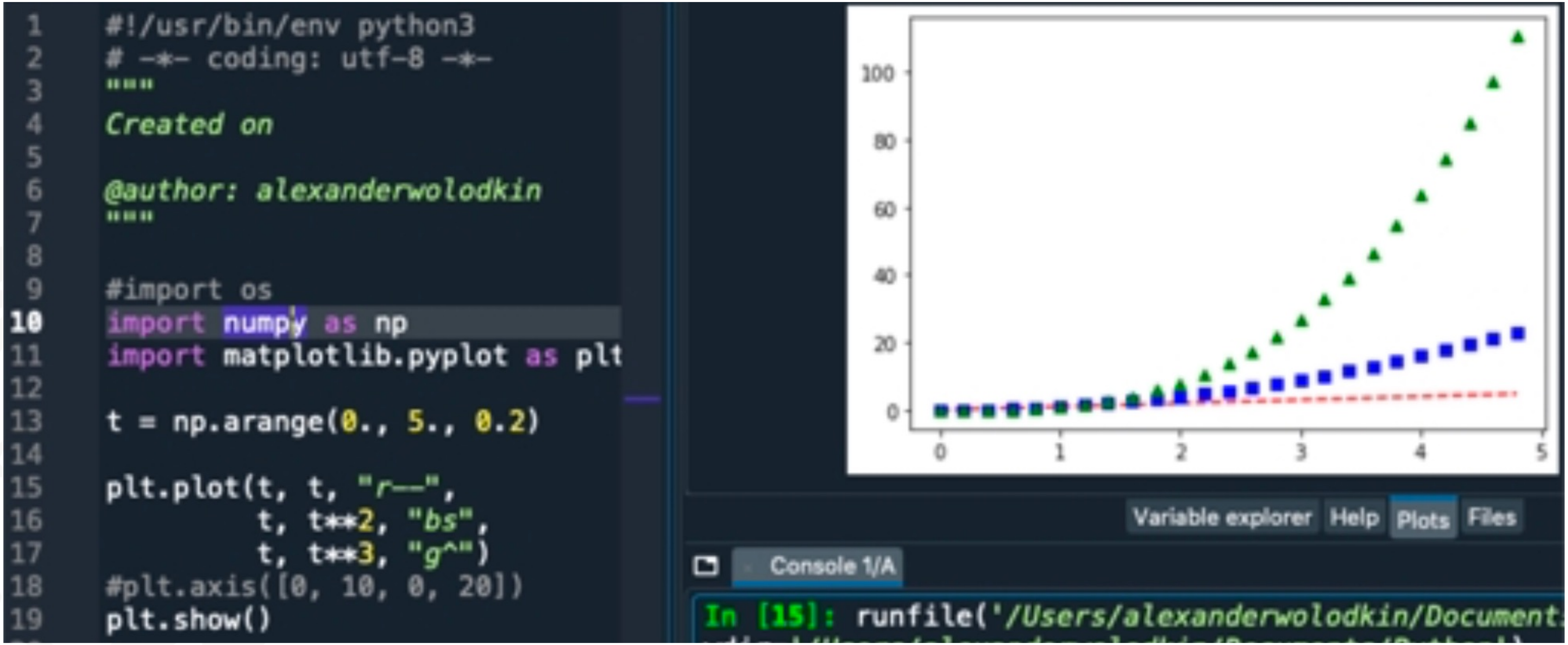
```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on

@author: alexanderwolodkin
"""

#import os
import matplotlib.pyplot as plt

plt.plot([1,2,3,4,5], "ro")
plt.plot([1,2,3],[1,4,9], "bx")
plt.show()
```

Basics of matplotlib



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

@author: alexanderwolodkin
"""


#import os
import matplotlib.pyplot as plt

plt.plot()
plt.axis([0, 10, 0, 20])
plt.show()
```

Basics of matplotlib



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

@author: alexanderwolodkin
"""

#import os
import numpy as np
import matplotlib.pyplot as plt

t = np.arange(0., 5., 0.2)

plt.plot(t, t, "r--",
         t, t**2, "bs",
         t, t**3, "g^")
#plt.axis([0, 10, 0, 20])
plt.show()
```

```
In [15]: runfile('/Users/alexanderwolodkin/Document
```