AI-based Audio Analysis of Music and Soundscapes

Fundamentals of Python Programming

Dr.-Ing. Jakob Abeßer

Fraunhofer IDMT

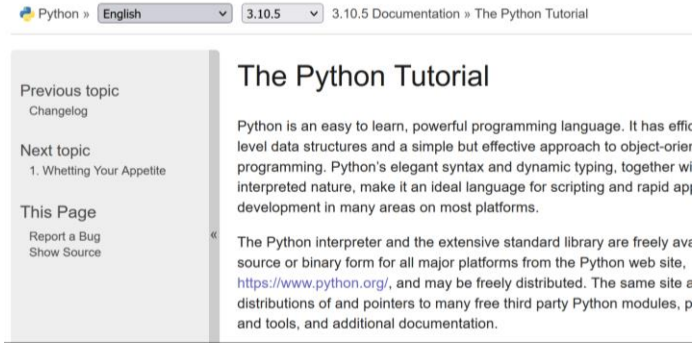
jakob.abesser@idmt.fraunhofer.de

Outline

- Python basics
- Data types
- NumPy (Numeric computing)
- Matplotlib (Data visualization)

Resources

- The Python Tutorial
  - https://docs.python.org/3/tutorial/


Python » English 3.10.5 3.10.5 Documentation » The Python Tutorial
Previous topic
Changelog
Next topic
1. Whetting Your Appetite
This Page
Report a Bug
Show Source
«

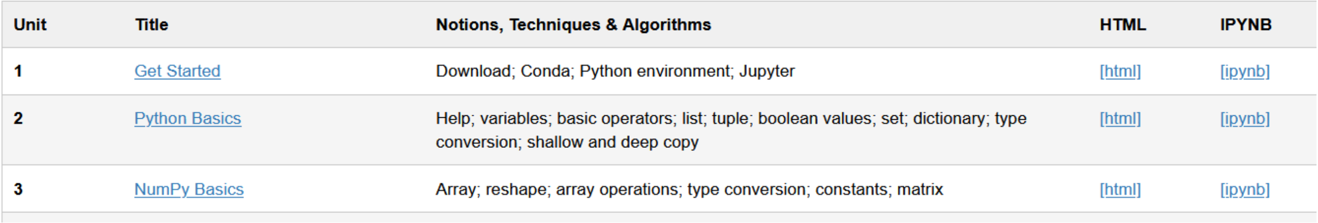
The Python Tutorial

Python is an easy to learn, powerful programming language. It has effic level data structures and a simple but effective approach to object-orier programming. Python's elegant syntax and dynamic typing, together wi interpreted nature, make it an ideal language for scripting and rapid ap development in many areas on most platforms.

The Python interpreter and the extensive standard library are freely ava source or binary form for all major platforms from the Python web site, https://www.python.org/, and may be freely distributed. The same site a distributions of and pointers to many free third party Python modules, p and tools, and additional documentation.

Fig. 1 - https://docs.python.org/3/tutorial/

- Preparation Course for Python
  - https://www.audiolabs-erlangen.de/resources/MIR/PCP/PCP.html

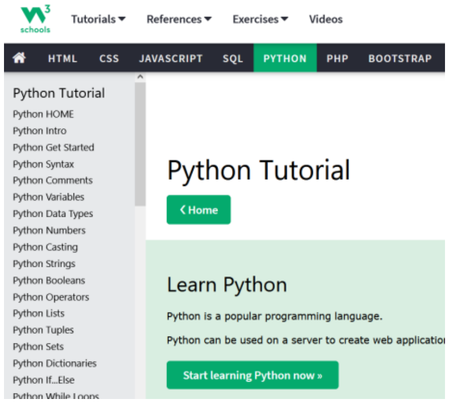


Unit	Title	Notions, Techniques & Algorithms	HTML	IPYNB
1	Get Started	Download; Conda; Python environment; Jupyter	[html]	[ipynb]
2	Python Basics	Help; variables; basic operators; list; tuple; boolean values; set; dictionary; type conversion; shallow and deep copy	[html]	[ipynb]
3	NumPy Basics	Array; reshape; array operations; type conversion; constants; matrix	[html]	[ipynb]

Fig. 2 - https://www.audiolabs-erlangen.de/resources/MIR/PCP/PCP.html

Resources

- W3 Schools – Python Tutorial
    - https://www.w3schools.com/python



Python Tutorial

Learn Python

Python is a popular programming language.

Python can be used on a server to create web applicatio

Fig. 3 - https://www.w3schools.com/python/

- Python Tutorial - Python Full Course for Beginners
    - https://www.youtube.com/watch?v=_uQrJ0TkZlc

Python Basics

- Free & simple to learn programming language (1989)
- Cross-platform (Windows, MacOS, Linux)
- Great for rapid prototyping
- Interpreted language (not compiled)

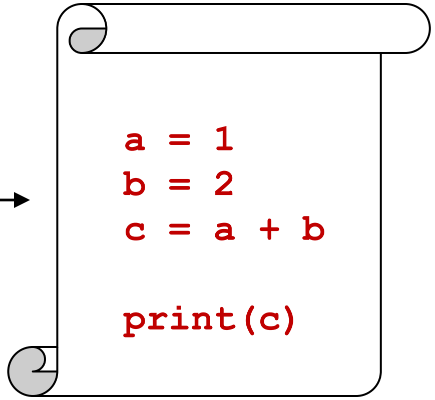
- Application Scenarios
  - Science
  - Web Development
  - Data Science / Data Visualization
  - Machine Learning / Artificial Intelligence
  - Desktop GUIs

Python Basics

Workflow

- Common workflow
  - Python Code-Files
  - Python interpreter

```
python myscript.py
```



```
a = 1
b = 2
c = a + b

print(c)
```

Python Basics

Indentations

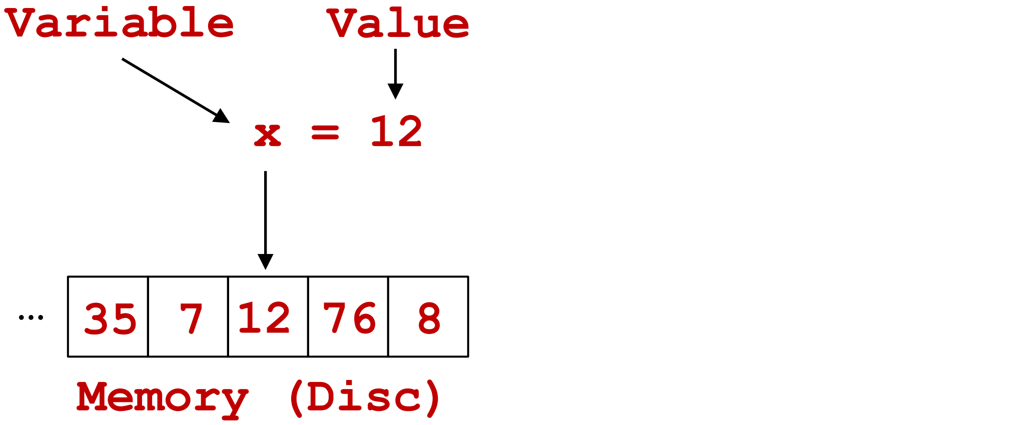
- Often “Tab” is used *(4 spaces are recommended)*
- Used to indicate block / level of code
    - Same number of spaces for the same level of code!

```
x = 12
If x > 24:
    print(x)
    if x > 32:
        print(">32")
```

Python Basics

Variables

- Variables
  - addresses a part of the **memory**
  - has a **name**
  - has a **value**



Python Basics

Variables

- Variables are not declared
- Variables are created after **value assignment**
- **Data type** is inferred from value

```
x = 12
print(x)   # 12

x = "Hello"
print(x)   # "Hello"
```

Python Basics
Variables

- Variable names can contain
  - Letters (a, b, c, …, A, B, C, …Z)
  - Underscore (_)
  - *(preferably, use small letters and underscore)*

```
first_result = 12.7

print(first_result)   # 12.7
```

Python Basics

Variables

- Access type

```
x = 12
print(x)         # 12
print(type(x))   # int
```

Python Basics

Comments

- One-line comments (#)

```
# this is a short note
```

- Multi-line comments (""")

```
"""
This is a longer comment
to explain more details.
"""
```

Python Basics

if/else & for-loops

- Conditional code execution

```
If a > 4:
    print("larger than four!")
else:
    print("smaller than four!")
```

- Iterate over list:

```
for i in range(4):
    print(i)

# 0, 1, 2, 3

for c in „yahoo":
    print(c)

# y, a, h, o, o
```

Python Basics

Functions

- Block of code (*one functionality*)
    - Name
    - Arguments

Argument(s)

```
def my_print(s):
    print(s)

my_print(123)    # 123
```

```
def my_addition(a, b):
    c = a + b
    return(c)
```

Return parameter

```
d = my_addition(1, 2)       # 3
e = my_addition(11, 22)     # 33
```

Python Basics

Functions

- Keyword arguments
  - Optional
  - Default values

Keyword argument(s)

```
def my_spectrogram(signal, db=True):
    # compute spectrogram …
    if db:
        # apply dB scaling
    # return spectrogram
```

Data Types

Strings

- Strings (text)

```
s = "Audio Analysis"
s = 'Audio Analysis`
s = str("Audio Analysis")
```

- Multiline strings

```
s = """Audio analysis
Is often based on signal
processing"""
```

Data Types

Strings

- Strings = Arrays (of bytes)

```
s = "Audio"
print(s[0])          # A
print(s[2])          # d
print(s[-1])         # o
```

- String length

```
s = "Audio"
print(len(s))        # 5
```

- Check for substring

```
s = "Hi Peter"
print("Hello" in s)    # False
print("Hi" in s)       # True
print("Hu" not in s)   # True
```

Data Types

Strings

- Slicing strings

```
s = "Audio"
print(s[0:2])        # Au
print(s[:2])         # Au
print(s[2:])         # dio
print(s[-2:])        # io
```

- Uppercase, Lowercase

```
s = "Audio"
print(s.upper())     # AUDIO
print(s.lower())     # audio
```

- Replace substring

```
s = "birdsong.wav"
s = s.replace(".wav", ".mp3")
print(s)             # birdsong.mp3
```

Data Types

Strings

- Splitting strings

```
s = "car.wav,12,BMW"
parts = s.split(",")
print(parts)
          # ['car.wav', '12', 'BMW']
```

- Joining strings

```
s = ["car","wav"]
filename = ".".join(s)
print(joint)      # car.wav
```

- Formatting strings

```
s1 = "{}.wav".format("car")
s2 = "car" + ".wav"
print(s1)            # car.wav
print(s2)            # car.wav
```

Data Types

Numeric Types

- Integers

```
i1 = 12
i2 = -23
```

- Float (floating point number)

```
f1 = 12.001
f2 = -23.5
```

- Type conversion

```
print(i1)               # 12
print(type(i1))         # int
i1 = float(i1)
print(i1)               # 12.0
print(type(i1))         # float
```

Data Types
Numeric Types

- Rounding up/down

```
import math
f = 1.49
print(math.ceil(f))    # 2
print(math.floor(f))   # 1
print(round(f))        # 1
```

Data Types
Lists

- Store data collections
  - Any data type

```
list1 = ["apple", "banana", "cherry"]

list2 = [1, 2, 3]

list3 = [True, False, True]
```

- Zero-based indexing

```
print(list2[0])      # 1
print(list1[2])      # Cherry
```

- Length

```
print(len(list1))    # 3
```

Data Types

Lists

- List comprehension

```
list1 = [2, 4, 6]

list2 = [_ + 1 for _ in list1]

# [3, 5, 7]
```

- Indexing / Slicing like for strings

Data Types

Dictionaries (dicts)

- Key/Value pairs

```
d = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

print(d["year"])    # 1964
```

- Keys

```
Print(d.keys())
    # ['brand', 'model', 'year']
```

Data Types

Operators

- Assignment operators

```
i = 12                 # 12
i = i + 2              # 14
i += 2                 # 16
i -= 2                 # 14
i /= 2                 # 7.0
```

- Comparison operators

```
print(1 == 1)          # True
print(1 <= 2)          # True
print(1 >= 3)          # False
print(1 != 3)          # True
```

Data Types

Operators

- Logical operators

```
i = 1
(i < 3) and (i > 1)    # False
(i < 3) or (i > 1)     # True
not (i > 5)            # True
```

NumPy (Numeric Computing)

- Standard library for working with numerical data in Python
- Core part of various Python libraries
  - Pandas (data analysis)
  - SciPy (scientific computing)
  - Matplotlib (visualization)
  - Scikit-learn (machine learning)
- Needs to be imported first

Alias *(for convenience)*

```
import numpy as np
```

NumPy

Arrays

- Efficient data structure to store multiple values (faster than lists)
- Contains
    - Raw data (values)
    - **dtype** (data type – np.int8 / np.float16 / np.float32)
    - **rank** (number of dimensions)
    - **shape** (size of array along each dimension

NumPy

Arrays

- Example (one-dimensional array)

```
a = np.array([1, 2, 3])
print(a)                  # [1, 2, 3]
print(a.ndim)             # 1
print(a.shape)            # (3,)
print(a.dtype)            # int32
```

1	2	3

- Example (two-dimensional array / matrix):

```
a = np.array([[1.1, 2.2], [3.3, 4.4]])
print(a)                  # [[1.1, 2.2]
                          #  [3.3, 4.4]]
print(a.ndim)             # 2
print(a.shape)            # (2,2)
print(a.dtype)            # float64
```

1.1	2.2
3.3	4.4

NumPy

Arrays

- Create arrays with ones / zeros

```
a = np.zeros([2, 3])
print(a)                   # array([[0., 0., 0.],
                           #        [0., 0., 0.]])

a = np.ones(3)
print(a)                   # array([1., 1., 1.])

a = np.ones(3, dtype=int)
print(a)                   # array([1, 1, 1])
```

NumPy

Arrays

- Create arrays with increasing numbers (**arange**)

```
a = np.arange(4)
print(a)                       # array([[0., 1., 2., 3.]])
```

- Indexing / Slicing nparray (like with lists & strings before)

```
a = np.arange(4)
print(a[0])                    # 0.
print(a[:2])                   # [0., 1.]
print(a[-1])                   # 3
```

NumPy

Arrays

- Concatenating two arrays (**concatenate**)

```
a = np.arange(4)
```

0	1	2	3

```
b = np.arange(3)
```

0	1	2

```
print(np.concatenate((a, b)))

                    # array([[0., 1., 2., 3., 0., 1., 2]])
```

0	1	2	3	0	1	2

NumPy
Arrays

- Horizontal stacking (**hstack**) and vertical stacking (**vstack**)

```
a = np.array((1, 2), dtype=int)
```

1	2

```
b = np.array((3, 4), dtype=int)
```

3	4

```
print(np.hstack((a, b)))    # array([1, 2, 3, 4])
```

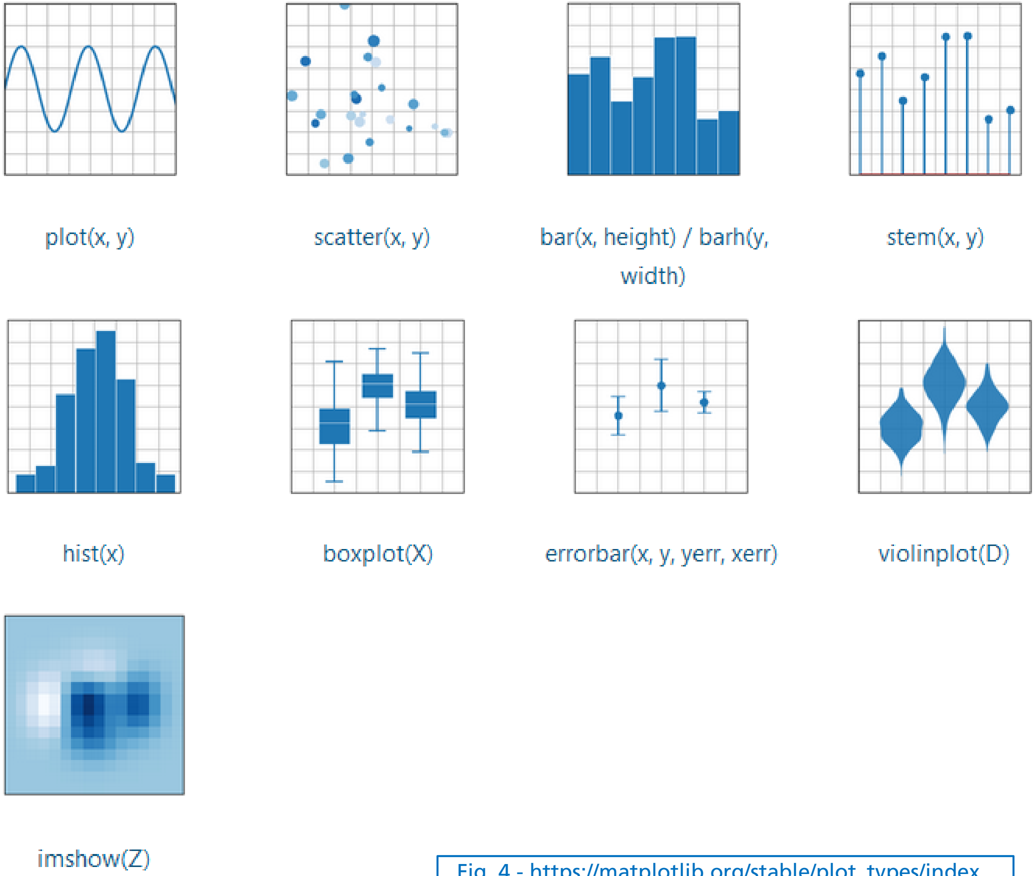
1	2	3	4

```
print(np.vstack((a, b)))    # array([[1, 2],
                                      [3, 4]])
```

1	2
3	4

Matplotlib (Data visualization)

- Plotting types



plot(x, y)

scatter(x, y)

bar(x, height) / barh(y, width)

stem(x, y)

hist(x)

boxplot(X)

errorbar(x, y, yerr, xerr)

violinplot(D)

imshow(Z)

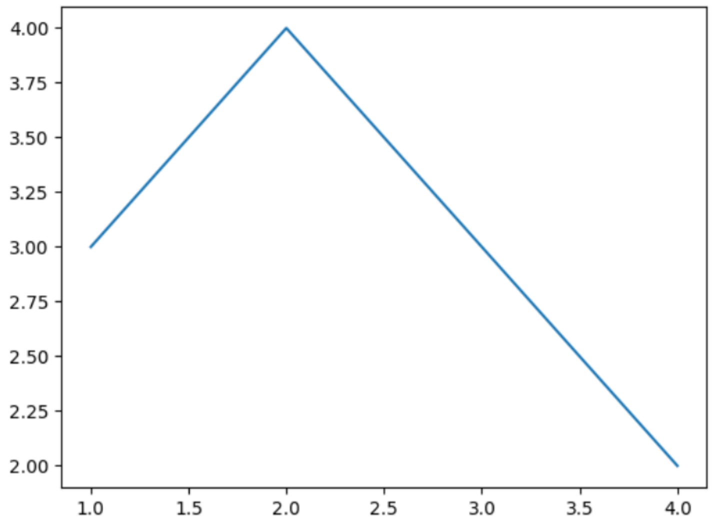
Fig. 4 - https://matplotlib.org/stable/plot_types/index

Matplotlib

First Steps

- Import matplotlib package
- Create figure
- Plot data & show figure

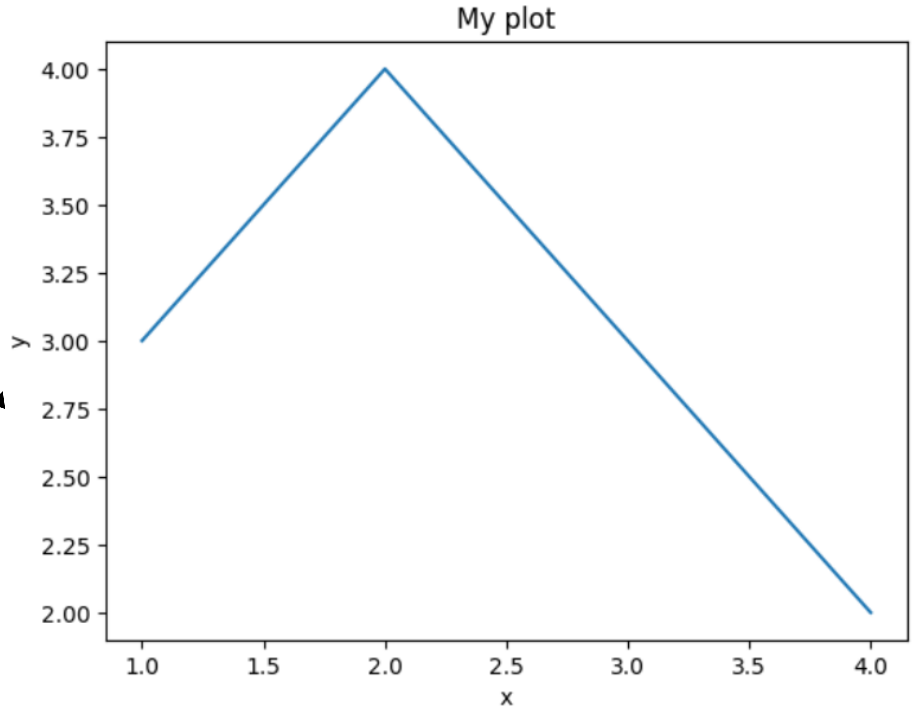
```
import numpy as np
x = np.array((1,2,3,4))
y = np.array((3,4,3,2))

import matplotlib.pyplot as plt

fig, ax = plt.subplots()

ax.plot(x, y)
plt.show()
```



Matplotlib

Axes Labels & Title

```
# ...

fig, ax = plt.subplots()

ax.plot(x, y)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_title('My plot‘)
plt.show()
```



Title

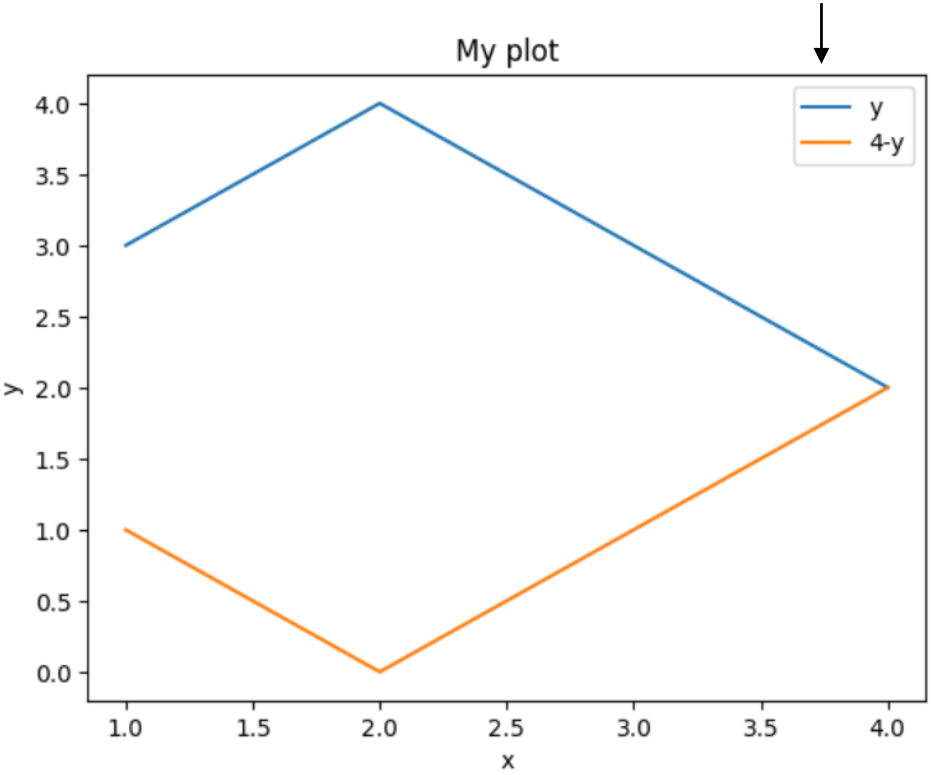
Axes Labels

Matplotlib

Legend

```
# ...

fig, ax = plt.subplots()

ax.plot(x, y, label='y')
ax.plot(x, 4-y, label='4-y‘)

# ...

plt.legend()
plt.show()
```

Legend



Matplotlib

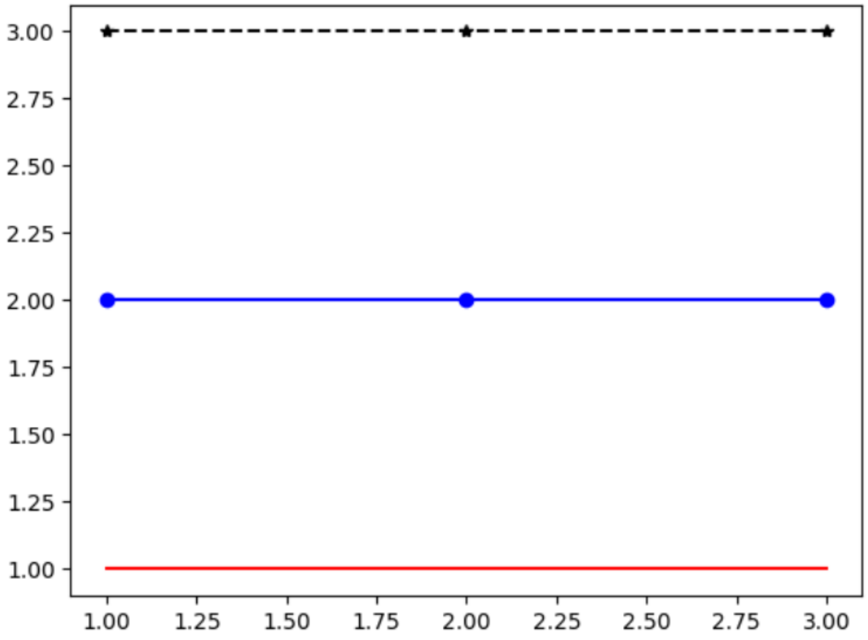
Line-style / marker-style

Color

Marker

Line-Style

```
fig, ax = plt.subplots()
ax.plot([1, 2, 3], [1, 1, 1], 'r-')
ax.plot([1, 2, 3], [2, 2, 2], 'bo-')
ax.plot([1, 2, 3], [3, 3, 3], 'k*--')
plt.show()
```



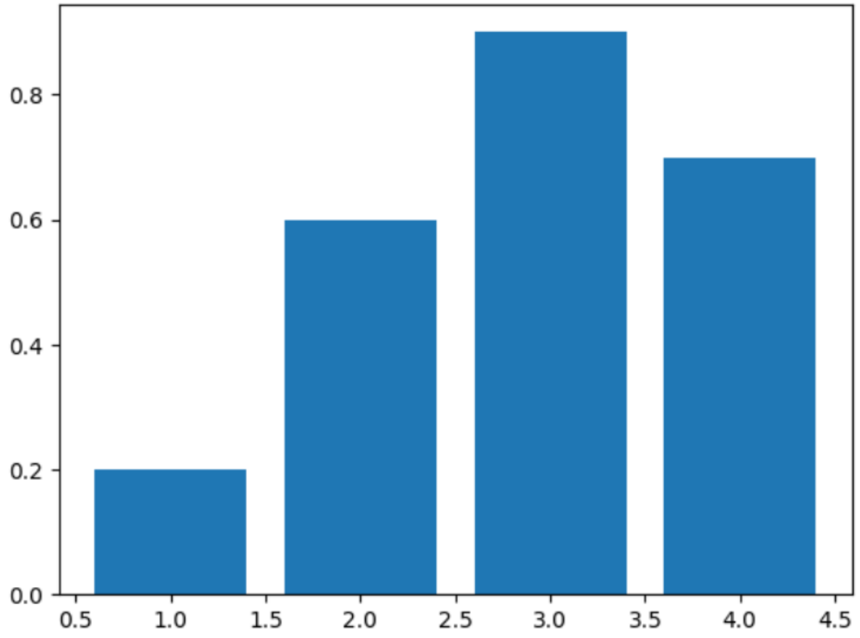
- Short or long form

```
...
..., 'k*--')
..., color='k', marker='*',
linestyle='--')
```

Matplotlib

Bar plot

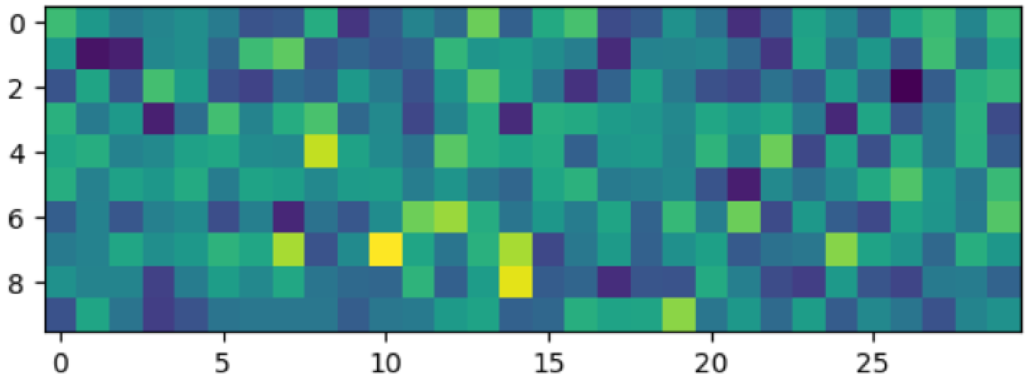
```
fig, ax = plt.subplots()
ax.bar([1, 2, 3, 4], [0.2, 0.6, 0.9, 0.7])
plt.show()
```



Matplotlib

Matrix plots

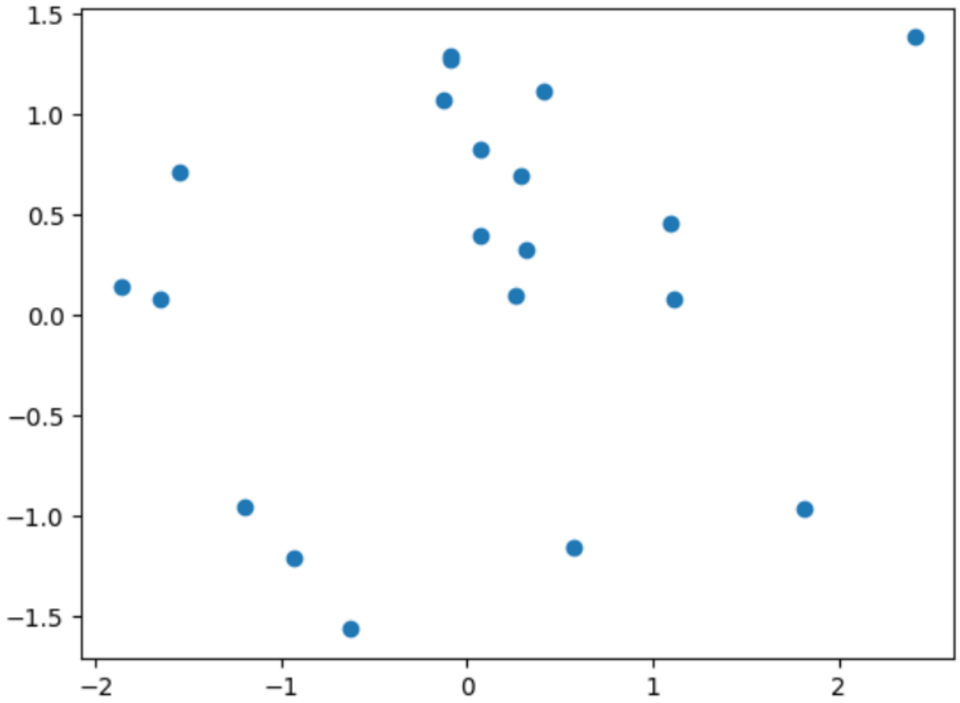
```
mat = np.random.randn(10, 30)
fig, ax = plt.subplots()
ax.imshow(mat)
plt.show()
```



Matplotlib

Scatter plots

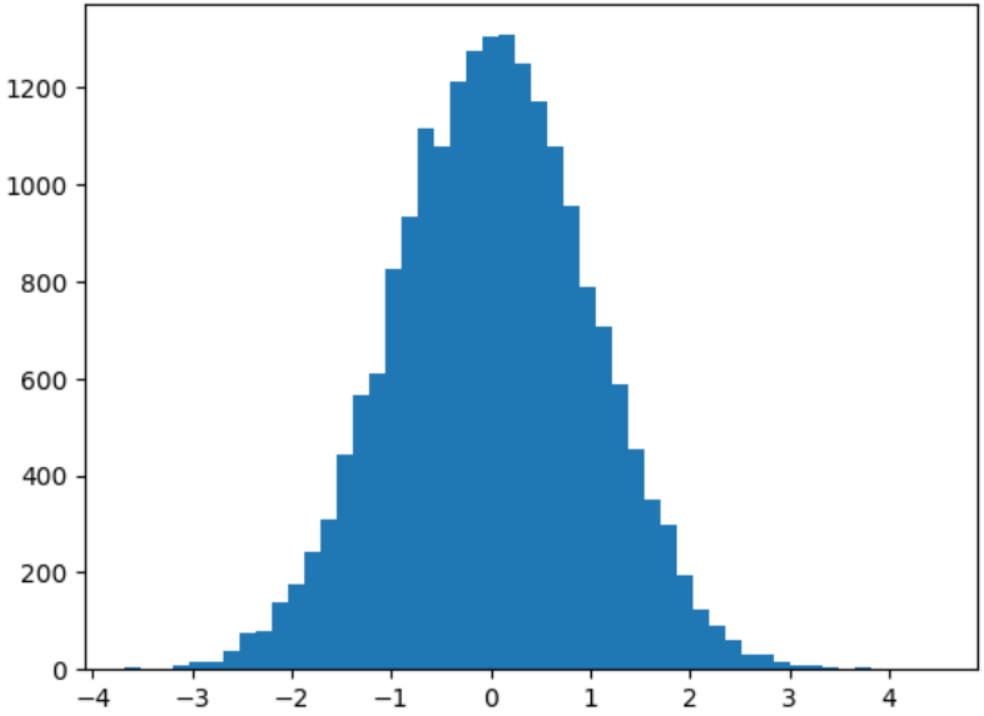
```
x = np.random.randn(20)
y = np.random.randn(20)
fig, ax = plt.subplots()
ax.scatter(x,y)
plt.show()
```



Matplotlib

Histograms

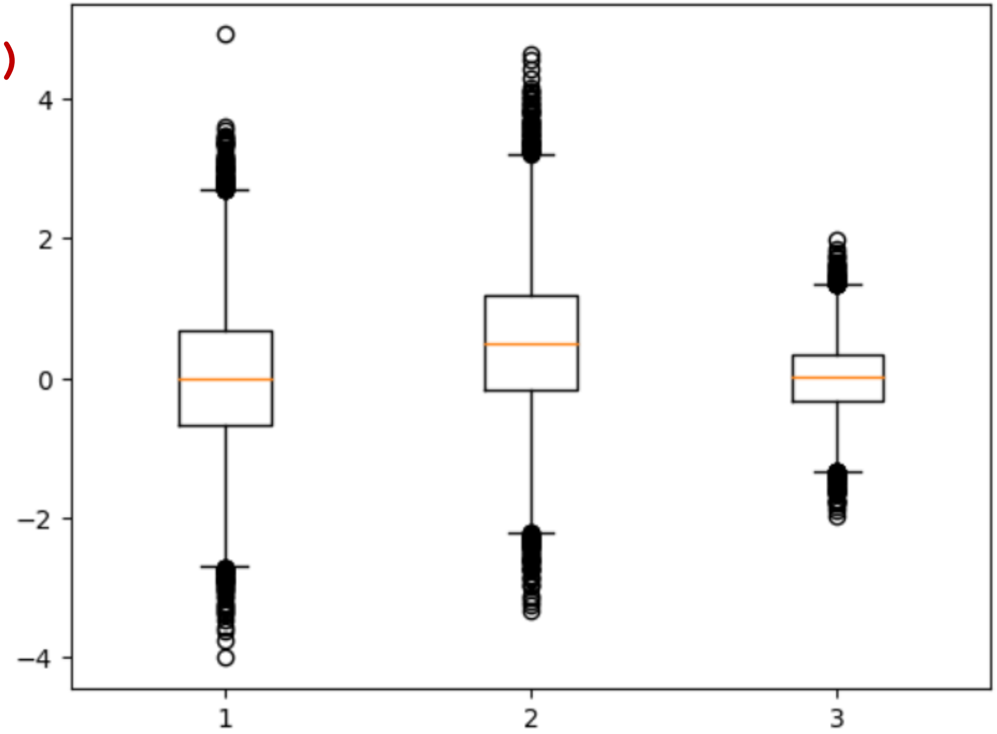
```
x = np.random.randn(20000)
fig, ax = plt.subplots()
ax.hist(x, bins=50)
plt.show()
```



Matplotlib

Boxplots

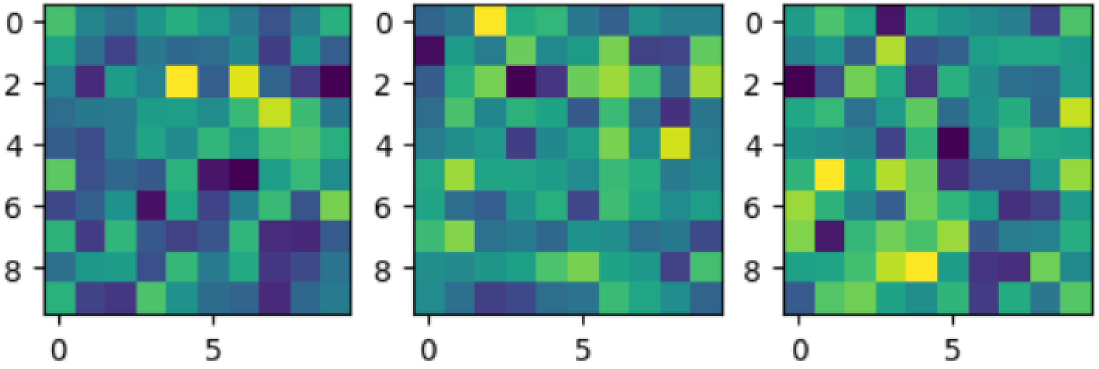
```
x = np.random.randn(20000, 3)
x[:, 1] += 0.5
x[:, 2] /= 2
fig, ax = plt.subplots()
ax.boxplot(x)
plt.show()
```



Matplotlib

Subplots

```
fig, ax = plt.subplots(1, 3)
ax[0].imshow(np.random.randn(10, 10))
ax[1].imshow(np.random.randn(10, 10))
ax[2].imshow(np.random.randn(10, 10))
plt.show()
```


0
2
4
6
8
0
5
0
2
4
6
8
0
5
0
2
4
6
8
0
5