

COP 3035

# Intro Programming in Python

Summer 2024

# Lecture 21 – part 1

Lab 10 (Optional)

Homework 7 – 07/29/24

Exam 4 – 08/02/24

# Lecture 21 – part 2

## Review

# Review

Integration Exercise

Python Modules

Python Packages

Create your own module

# Python Modules

- A **module** is a file containing Python definitions and statements.
- The file name is the module name with the suffix **.py** added.
- Modules are used to organize code logically by grouping related functions, classes, and variables. This makes the code easier to understand and use.
- Modules provide their own namespaces, which helps avoid naming conflicts between identifiers.



`module_name.py`

Basic Import syntax:

```
import module_name
```

Selective Import syntax:

```
from module_name import function_name
```

Alias Import syntax:

```
import module_name as mn
```

```
from module_name import function_name as fn
```

# Creating Your Own Modules

- Simply save your code in a **.py** file.
- This file can then be imported into other Python scripts.
- Use the dot notation (**module\_name.function\_name**) to access functions and variables defined in the module.
- Use docstrings (**""" text """** ) to document the module, classes and functions.

# The Python Standard Library

<https://docs.python.org/3/library/index.html>

Python comes with a rich standard library, which is a collection of modules that provides access to system functionality and standardized solutions.

Module	Description
<b>os</b>	Offers functions to interact with the operating system, such as file and directory operations, executing commands, others.
<b>sys</b>	Provides access to some variables and functions that interact with the Python interpreter, allowing manipulation of the runtime environment.
<b>datetime</b>	For manipulating dates and times, calculating differences, and formatting.
<b>math</b>	Mathematical functions, including trigonometric, logarithmic, and more.
<b>random</b>	Used for generating pseudo-random numbers for various distributions and choosing randomly from sequences.
<b>json</b>	Supports encoding and decoding JSON data, crucial for web data interchange and configuration files.
<b>re</b>	Supports regular expressions for advanced string manipulation and pattern matching.

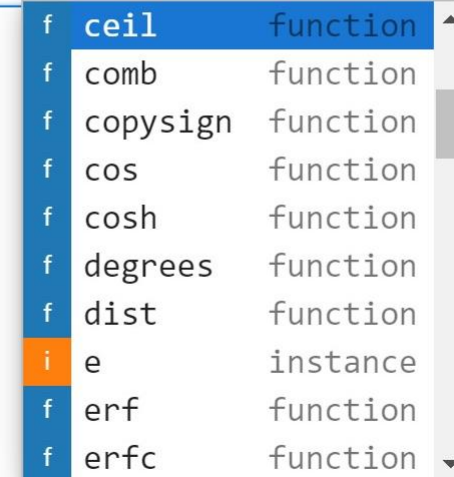
Example:

```
[1]: import math
```

```
[3]: math.factorial(4)
```

```
[3]: 24
```

```
[ ]: math.
```



f	ceil	function
f	comb	function
f	copysign	function
f	cos	function
f	cosh	function
f	degrees	function
f	dist	function
i	e	instance
f	erf	function
f	erfc	function

# Python packages

A module is a single file containing Python code, whereas a package is a collection of modules that are organized in a directory hierarchy.

Parameter	Module	Package
Definition	It can be a simple Python file (.py extension) that contains collections of functions and global variables.	A package is a collection of different modules with an <code>__init__.py</code> file.
Purpose	Code organization	Code distribution and reuse
Organization	Code within a single file	Related modules in a directory hierarchy
Sub-modules	None	Multiple sub-modules and sub-packages
Required Files	Only Python file (.py format)	<code>__init__.py</code> file and additional Python files
How to Import	<code>import module_name</code>	<code>import package_name.module_name</code> or <code>from package_name import module_name</code>
Example	math, random, os, datetime, csv	Numpy, Pandas, Matplotlib, django



# Lecture 21 – part 3

## Matplotlib

## Matplotlib:

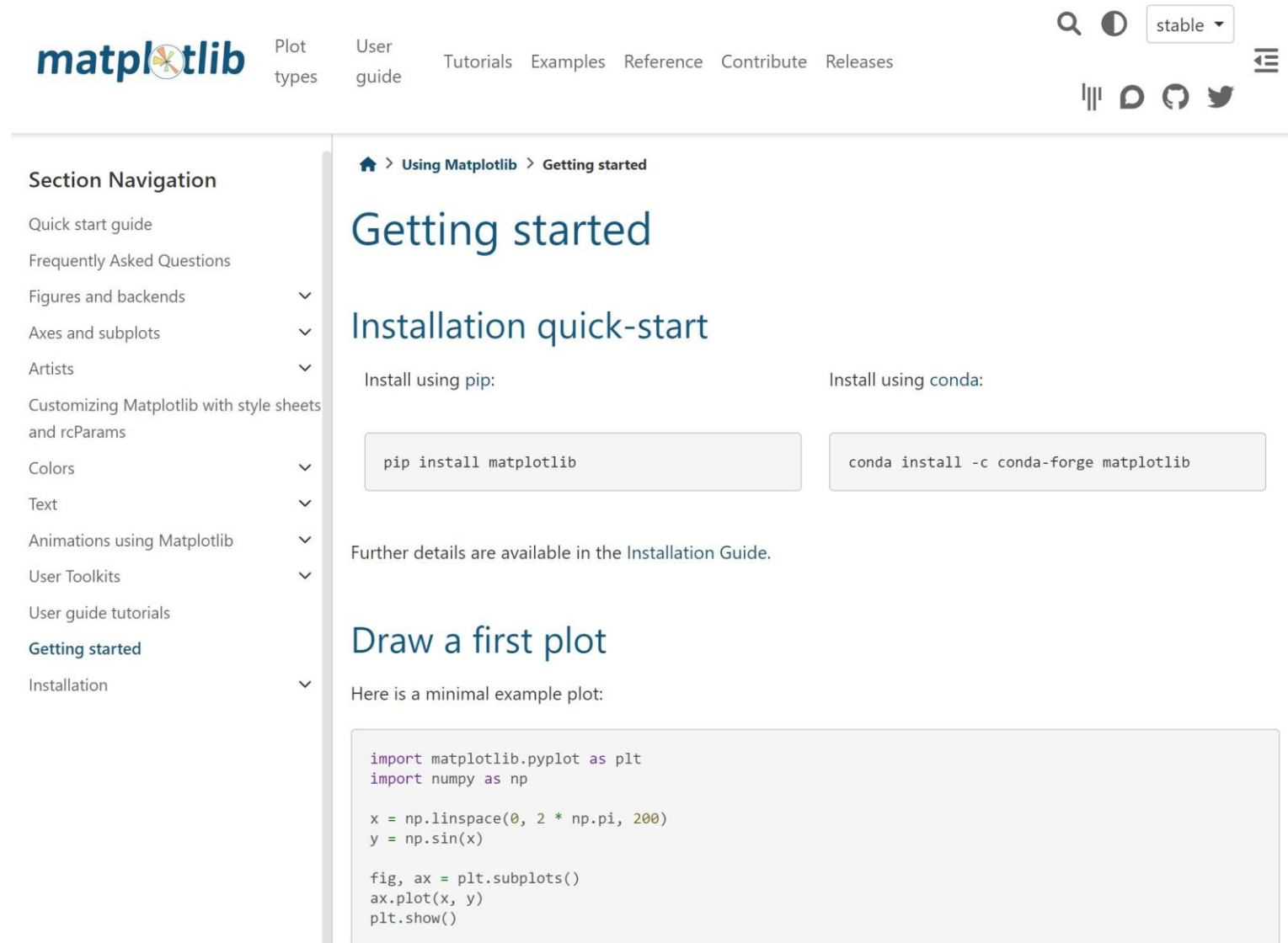
A comprehensive library for creating static, animated, and interactive visualizations in Python.

## Installation:

```
pip install matplotlib
```

## Importing:

```
import matplotlib.pyplot as plt
```



The screenshot shows the Matplotlib website's 'Getting started' page. The top navigation bar includes the Matplotlib logo, links for 'Plot types', 'User guide', 'Tutorials', 'Examples', 'Reference', 'Contribute', and 'Releases'. A search bar and a 'stable' version selector are on the right. The left sidebar contains a 'Section Navigation' menu with links to 'Quick start guide', 'Frequently Asked Questions', 'Figures and backends', 'Axes and subplots', 'Artists', 'Customizing Matplotlib with style sheets and rcParams', 'Colors', 'Text', 'Animations using Matplotlib', 'User Toolkits', 'User guide tutorials', 'Getting started' (highlighted), and 'Installation'. The main content area is titled 'Getting started' and 'Installation quick-start'. It provides instructions for installing Matplotlib using pip and conda, with corresponding code snippets in light blue boxes. The pip snippet is 'pip install matplotlib' and the conda snippet is 'conda install -c conda-forge matplotlib'. Below the installation instructions, it states 'Further details are available in the Installation Guide.' and 'Draw a first plot'. A minimal example plot code snippet is provided in a light blue box: 

```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 2 * np.pi, 200)
y = np.sin(x)

fig, ax = plt.subplots()
ax.plot(x, y)
plt.show()
```

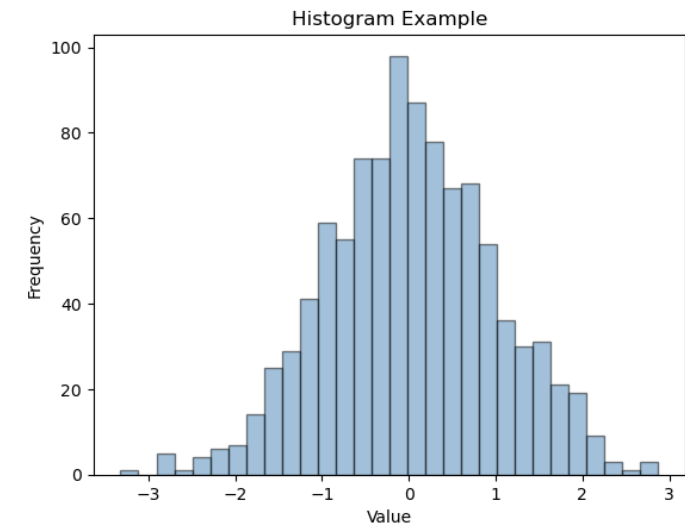
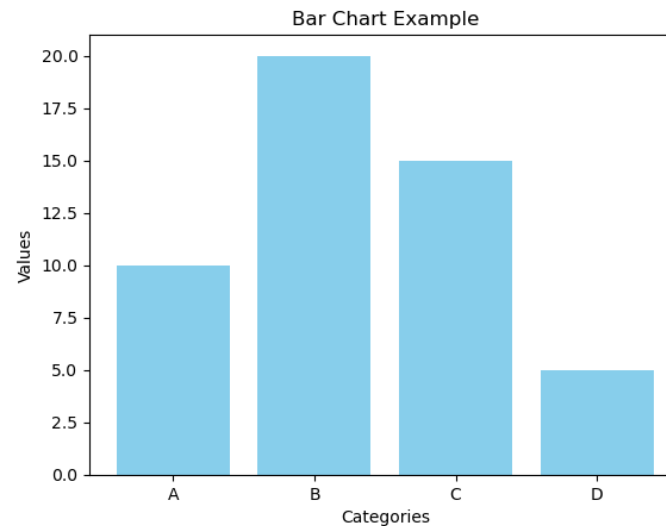
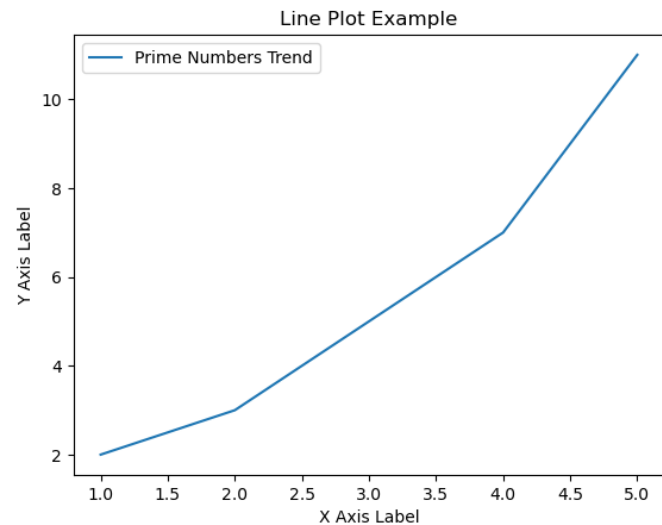
[https://matplotlib.org/stable/users/getting\\_started/](https://matplotlib.org/stable/users/getting_started/)

# Basic plotting

**Line Plot :** Basic syntax and customization (color, linestyle, marker).

**Bar Chart:** Comparing data side-by-side.

**Histogram:** Visualizing distributions.



[https://matplotlib.org/stable/users/explain/quick\\_start.html](https://matplotlib.org/stable/users/explain/quick_start.html)

# matplotlib.pyplot.plot()

The matplotlib.pyplot.plot() function is quite flexible, and its syntax can vary depending on how you want to customize your plot.

Syntax:

```
plot([x], y, [fmt], *, data=None, **kwargs)
plot([x], y, [fmt], [x2], y2, [fmt2], ..., **kwargs)
```

**x, y:** These are arrays or sequences of values. x is optional; if not provided, the default will be range(len(y)).

**fmt:** A format string, optional, that specifies color, marker, and line type in a shorthand form. For example, 'ro-' means red circles connected by lines.

NOTE: The **\*\*kwargs** parameter lets you pass in numerous other options to customize markers, lines, and more.

## Common Parameters

<b>data:</b>	An optional parameter that allows specifying the data source (Dictionary).
<b>color:</b>	Specifies the color of the line. You can use named colors, hex codes, or RGB/A tuples.
<b>label:</b>	Sets the label for this line, which will appear in the legend.
<b>linewidth</b> or <b>lw:</b>	Sets the width of the line.
<b>markersize</b> or <b>ms:</b>	Determines the size of the markers.
<b>linestyle</b> or <b>ls:</b>	Defines the style of the line, such as solid, dashed, or none.
<b>marker:</b>	Chooses the marker style for the points, like circles, squares, etc.

```
plt.plot(y)      # Assumes x as range(len(y))
plt.plot(x, y)   # Plots y vs x
plt.plot(x, y, 'bo-') # Blue circles with solid lines
plt.plot(x, y, color='green', marker='o', linestyle='dashed',
         linewidth=2, markersize=12)
```

**Line Styles**

Symbol	Name
-	Solid
--	Dashed
-.	Dash-dot
:	Dotted
None	No line
"	No line

**Markers**

Symbol	Name
.	Point
,	Pixel
o	Circle
v	Triangle Down
^	Triangle Up
<	Triangle Left
>	Triangle Right
1	Tri Down
2	Tri Up
3	Tri Left
4	Tri Right

Symbol	Name
8	Octagon
s	Square
p	Pentagon
*	Star
h	Hexagon1
H	Hexagon2
+	Plus
x	X
D	Diamond
d	Thin Diamond
`	`
—	Hline

color = 'cyan'  
color='c'  
color='#00FFFF'  
color=(0,1,1)

## Color

Color	Short Name	Hex Code	RGB Tuple (*)
Black	k	#000000	(0, 0, 0)
White	w	#FFFFFF	(255, 255, 255)
Red	r	#FF0000	(255, 0, 0)
Green	g	#008000	(0, 128, 0)
Blue	b	#0000FF	(0, 0, 255)
Cyan	c	#00FFFF	(0, 255, 255)
Magenta	m	#FF00FF	(255, 0, 255)
Yellow	y	#FFFF00	(255, 255, 0)
Light Blue	-	#ADD8E6	(173, 216, 230)
Orange	-	#FFA500	(255, 165, 0)
Purple	-	#800080	(128, 0, 128)
Brown	-	#A52A2A	(165, 42, 42)
Pink	-	#FFC0CB	(255, 192, 203)
Gray	-	#808080	(128, 128, 128)
Lime	-	#00FF00	(0, 255, 0)

\* Note: You need to convert values to [0,1] by dividing by 255.

## Example using the data parameter:

```
import matplotlib.pyplot as plt

# Sample data as a dictionary
data = {
    'x': range(1, 11),
    'y1': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    'y2': [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
}

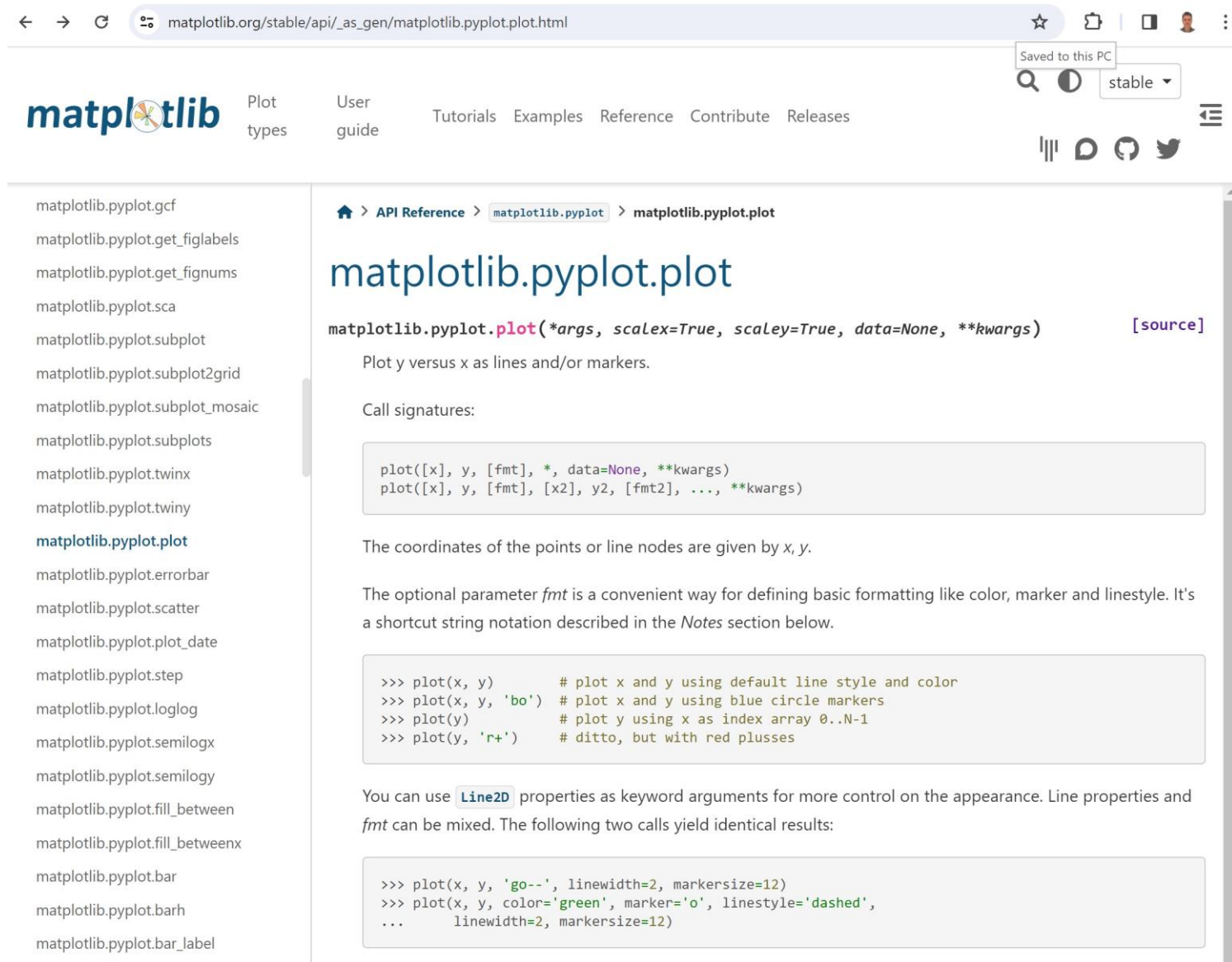
# Plotting with explicit formatting
plt.plot('x', 'y1', data=data, marker='o', color='red', linestyle='-', label='Ascending')
plt.plot('x', 'y2', data=data, marker='^', color='blue', linestyle='--', label='Descending')

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



## API Reference:

[https://matplotlib.org/stable/api/\\_as\\_gen/matplotlib.pyplot.plot.html](https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.plot.html)



The screenshot shows a web browser displaying the matplotlib API reference page for `matplotlib.pyplot.plot`. The browser's address bar shows the URL `matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.plot.html`. The page features a navigation bar with the matplotlib logo and links to Plot types, User guide, Tutorials, Examples, Reference, Contribute, and Releases. A sidebar on the left lists various plotting functions, with `matplotlib.pyplot.plot` highlighted. The main content area shows the function signature `matplotlib.pyplot.plot(*args, scalex=True, scaley=True, data=None, **kwargs)` with a [source] link. Below the signature, it states "Plot y versus x as lines and/or markers." and "Call signatures:". A code block shows two examples of plot calls. The text explains that the coordinates of the points or line nodes are given by `x, y`. It also mentions the optional parameter `fmt` for defining basic formatting like color, marker, and linestyle. Another code block shows two more examples of plot calls using `Line2D` properties as keyword arguments. The page is styled with a clean, modern design, using a light gray background and blue accents for links and highlights.

matplotlib.org/stable/api/\_as\_gen/matplotlib.pyplot.plot.html

matplotlib

Plot types User guide Tutorials Examples Reference Contribute Releases

matplotlib.pyplot.plot

matplotlib.pyplot.plot(\*args, scalex=True, scaley=True, data=None, \*\*kwargs) [source]

Plot y versus x as lines and/or markers.

Call signatures:

```
plot([x], y, [fmt], *, data=None, **kwargs)
plot([x], y, [fmt], [x2], y2, [fmt2], ..., **kwargs)
```

The coordinates of the points or line nodes are given by `x, y`.

The optional parameter `fmt` is a convenient way for defining basic formatting like color, marker and linestyle. It's a shortcut string notation described in the *Notes* section below.

```
>>> plot(x, y)           # plot x and y using default line style and color
>>> plot(x, y, 'bo')      # plot x and y using blue circle markers
>>> plot(y)              # plot y using x as index array 0..N-1
>>> plot(y, 'r+')         # ditto, but with red plusses
```

You can use `Line2D` properties as keyword arguments for more control on the appearance. Line properties and `fmt` can be mixed. The following two calls yield identical results:

```
>>> plot(x, y, 'go--', linewidth=2, markersize=12)
>>> plot(x, y, color='green', marker='o', linestyle='dashed',
...      linewidth=2, markersize=12)
```

# Exercises

1. Draw a plot of a simple list [1,2,3,4,5]. Then add a title and a grid.
2. Draw a plot using two lists X and Y. Add title, grid, axis titles and a line label.
3. Draw three plots for  $y_1=x$ ,  $y_2=x^2$  and  $y_3=x^3$ , for the first 50 numbers. Use different colors and markers.
4. Generate a line plot with three different mathematical functions:  
     $y_1 = \sin(x)$  ,  
     $y_2 = \cos(x)$  and  
     $y_3 = 2\sin(x)\cos(x)$ ,  
    for x ranging from 0 to  $2\pi$ .  
    Use different line styles and colors for each function. Include a title and legends. Label the axis.
5. Complete the tutorial for matplotlib.pyplot at:  
<https://matplotlib.org/stable/tutorials/pyplot.html#sphx-glr-tutorials-pyplot-py>

# Lecture 21 – part 4

NumPy

# What is NumPy ?

- **NumPy** stands for Numerical Python.
- It's a library for Python that supports **large, multi-dimensional arrays and matrices**.
- Provides a wide range of **mathematical functions** to operate on these arrays **efficiently**.
- **Simplifies** complex numerical operations, making code more readable and concise.
- Works well with **other libraries** in the Python data ecosystem (e.g., pandas, matplotlib).
- **Common Use Cases:** Data analysis, Machine Learning, Image processing, simulations

## Installation:

```
pip install numpy
```

## Importing:

```
import numpy as np
```

## Creating an array:

```
arr = np.array([1, 2, 3])
```

NumPy's array class is called **ndarray**



User  
Guide

API  
reference

Building from  
source

Development

Release  
notes

Learn ↗ More ▾



devdocs ▾



## Getting started

What is NumPy?

Installation ↗

## NumPy quickstart

NumPy: the absolute basics for beginners

## Fundamentals and usage

NumPy fundamentals

NumPy for MATLAB users

NumPy tutorials ↗

NumPy how-tos

## Advanced usage and interoperability

Using NumPy C-API

F2PY user guide and reference manual

Under-the-hood documentation for developers

Interoperability with NumPy

## Extras

Glossary

# The basics

NumPy's main object is the homogeneous multidimensional array. It is a table of elements (usually numbers), all of the same type, indexed by a tuple of non-negative integers. In NumPy dimensions are called *axes*.

For example, the array for the coordinates of a point in 3D space, `[1, 2, 1]`, has one axis. That axis has 3 elements in it, so we say it has a length of 3. In the example pictured below, the array has 2 axes. The first axis has a length of 2, the second axis has a length of 3.

```
[[1., 0., 0.],  
 [0., 1., 2.]]
```

NumPy's array class is called `ndarray`. It is also known by the alias `array`. Note that `numpy.array` is not the same as the Standard Python Library class `array.array`, which only handles one-dimensional arrays and offers less functionality. The more important attributes of an `ndarray` object are:

### `ndarray.ndim`

the number of axes (dimensions) of the array.

### `ndarray.shape`

the dimensions of the array. This is a tuple of integers indicating the size of the array in each dimension. For a matrix with  $n$  rows and  $m$  columns, `shape` will be `(n,m)`. The length of the `shape` tuple is therefore the number of axes, `ndim`.

### `ndarray.size`

the total number of elements of the array. This is equal to the product of the elements of `shape`.

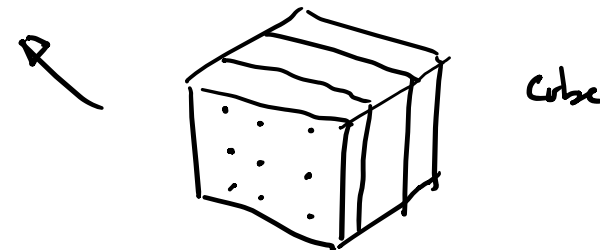
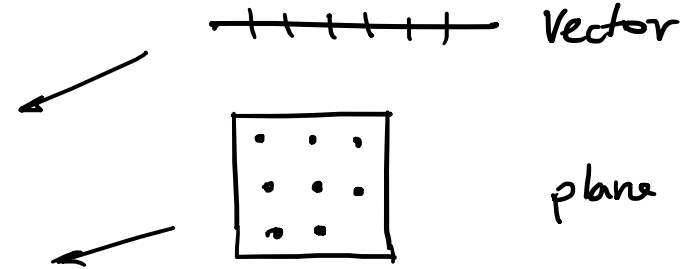
# NumPy dimensions

- NumPy dimensions are called **axes**

```
one_d_array = np.array([1, 2, 3, 4, 5])
```

```
two_d_array = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
```

```
three_d_array = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]], [[13, 14, 15], [16, 17, 18]]])
```



# Basics

Feature	Description	Example Code
<code>np.array</code>	Creates an array.	<code>A = np.array([1, 2, 3])</code>
<code>.ndim</code>	Number of array dimensions.	<code>A.ndim</code>
<code>.shape</code>	Dimensions of the array.	<code>A.shape</code>
<code>.size</code>	Number of elements in the array.	<code>A.size</code>
<code>np.zeros</code>	Creates an array filled with zeros.	<code>np.zeros((2, 3))</code>
<code>np.ones</code>	Creates an array filled with ones.	<code>np.ones((3, 3))</code>
<code>np.arange</code>	Creates an array with a range of values.	<code>np.arange(0, 10, 2)</code>
<code>np.linspace</code>	Creates an array with evenly spaced values.	<code>np.linspace(0, 1, 5)</code>

# Operations

Feature	Description	Example Code
-	Subtraction element-wise.	<code>A - B</code>
**	Exponentiation element-wise.	<code>A ** 2</code>
*	Multiplication element-wise.	<code>A * B</code>
<	Element-wise comparison.	<code>A &lt; B</code>
.dot()	Dot product of two arrays.	<code>A.dot(B)</code> or <code>np.dot(A, B)</code>



# Math

Feature	Description	Example Code
<code>np.add()</code>	Element-wise addition.	<code>np.add(A, B)</code>
<code>np.exp()</code>	Exponential of all elements.	<code>np.exp(A)</code>
<code>np.sqrt()</code>	Square root of each element.	<code>np.sqrt(A)</code>
<code>np.sin()</code>	Sine of each element.	<code>np.sin(A)</code>
<code>np.cos()</code>	Cosine of each element.	<code>np.cos(A)</code>

# Indexing and Slicing

Feature	Description	Example Code
Single element indexing	Access a single element by its position.	<code>arr[2]</code> or <code>arr[2, 3]</code>
Slice along one dimension	Select a range of elements from an array.	<code>arr[0:5]</code> or <code>arr[:5]</code>
Slice along two dimensions	Select a rectangle of elements.	<code>arr[1:4, 0:3]</code>
Stride for slicing	Select elements with a step size between them.	<code>arr[:, 2]</code> or <code>arr[0:5:2]</code>
Negative slicing	Use negative indices to slice from the end of the array.	<code>arr[-3:]</code> or <code>arr[1:-1]</code>
Boolean indexing	Select elements based on a boolean condition.	<code>arr[arr &gt; 5]</code>
Fancy indexing	Index with integer arrays.	<code>arr[[1, 3, 4]]</code> or <code>arr[[1, 2], [3, 4]]</code>
Mixing integer and slice	Combine integer indexing and slicing.	<code>arr[1, :2]</code> or <code>arr[2:3, :1]</code>
Ellipsis (...)	Used to replace multiple colons.	<code>arr[..., 1]</code> (same as <code>arr[:, 1]</code> )

# NumPy Array Manipulations

## **.ravel()**

- Flattens an array into a contiguous 1D array.
- Returns a view of the original array whenever possible, making it memory efficient.
- Changes to the returned array may affect the original array.
- Example: **flattened\_array = arr.ravel()**

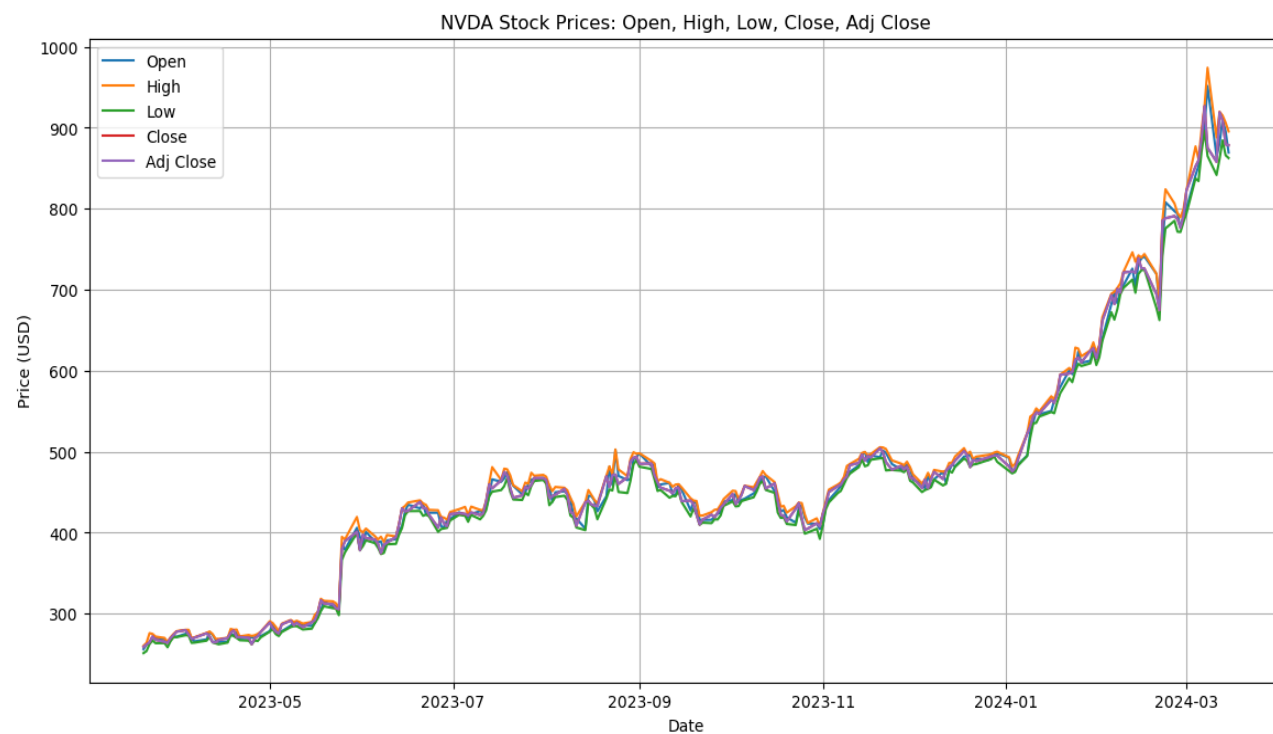
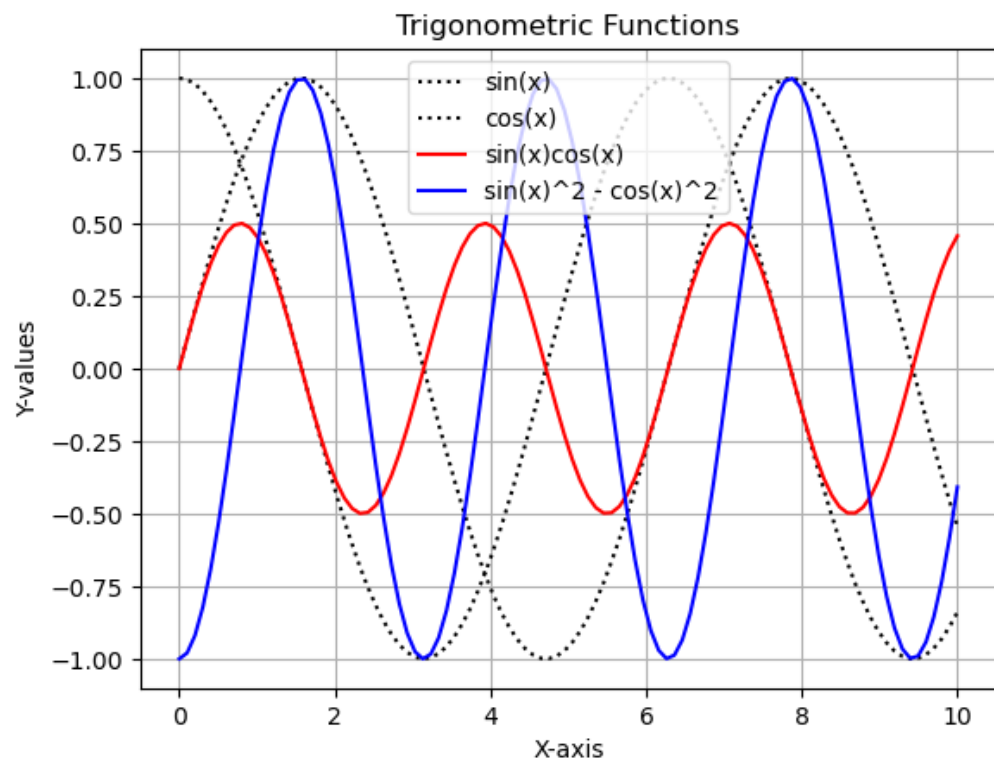
## **.reshape()**

- Gives a new shape to an array without changing its data.
- Returns a new array with the specified shape.
- Can return a view or a copy, depending on the memory layout.
- Use -1 to automatically calculate the size of one dimension.
- Example: **reshaped\_array = arr.reshape(2, 6)**

## **.resize()**

- Alters the size and shape of an array in-place.
- Can expand or shrink the array; new elements are filled with zeros.
- Does not return a value; modifies the original array.
- Example: **arr.resize((2, 6))**

## Example: Using Numpy, Matplotlib and Pandas



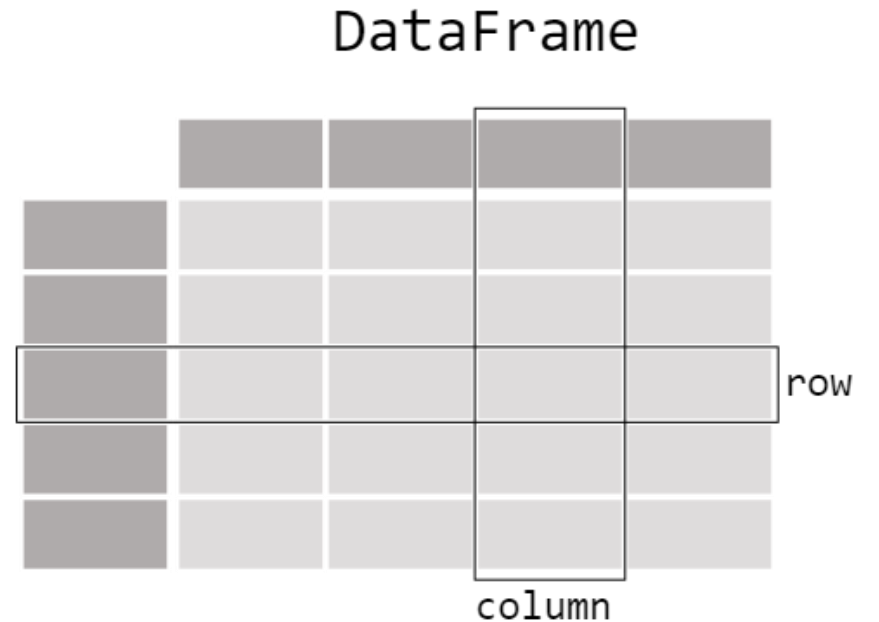
# Lecture 21 – part 5

## Pandas

# Pandas

Pandas is a fast, powerful, flexible and easy to use open-source data analysis and manipulation tool, built on top of the Python programming language.

- Provides essential data structures like **DataFrame** and **Series** for efficient data manipulation and analysis.
- Supports **various file formats** including CSV, Excel, and SQL, facilitating easy data reading and writing.
- Offers **tools** for handling missing data, merging, joining, and filtering datasets effectively.
- Includes **robust features for time series** data manipulation such as resampling and frequency conversion.
- Backed by a **strong community** with extensive documentation, making it a reliable choice for data scientists and analysts.



## Installation

```
pip install pandas
```

## Import

```
import pandas as pd
```

The screenshot shows a web browser displaying the pandas documentation page titled "10 minutes to pandas". The browser's address bar shows the URL "pandas.pydata.org/docs/user\_guide/10min.html". The page features a navigation bar with links to "Getting started", "User Guide" (which is highlighted), "API reference", "Development", and "Release notes". A search bar and a version selector set to "2.2 (stable)" are also present. On the left, a sidebar lists various topics, with "10 minutes to pandas" selected. The main content area has a breadcrumb trail "User Guide > 10 minutes to pandas" and a large heading "10 minutes to pandas". Below the heading, a paragraph introduces the tutorial as a short introduction for new users, with a link to the "Cookbook". A code block shows the standard imports for pandas: 

```
In [1]: import numpy as np
In [2]: import pandas as pd
```

. The next section is titled "Basic data structures in pandas" and explains that pandas provides two types of classes for handling data. It lists: 1. **Series**: a one-dimensional labeled array holding data of any type such as integers, strings, Python objects etc. 2. **DataFrame**: a two-dimensional data structure that holds data like a two-dimension array or a table with rows and columns.

10 minutes to pandas

Intro to data structures

Essential basic functionality

IO tools (text, CSV, HDF5, ...)

PyArrow Functionality

Indexing and selecting data

MultilIndex / advanced indexing

Copy-on-Write (CoW)

Merge, join, concatenate and compare

Reshaping and pivot tables

Working with text data

Working with missing data

Duplicate Labels

Categorical data

Nullable integer data type

Nullable Boolean data type

Chart visualization

Table Visualization

10 minutes to pandas

This is a short introduction to pandas, geared mainly for new users. You can see more complex recipes in the [Cookbook](#).

Customarily, we import as follows:

```
In [1]: import numpy as np
In [2]: import pandas as pd
```

## Basic data structures in pandas

Pandas provides two types of classes for handling data:

1. **Series**: a one-dimensional labeled array holding data of any type such as integers, strings, Python objects etc.
2. **DataFrame**: a two-dimensional data structure that holds data like a two-dimension array or a table with rows and columns.

[https://pandas.pydata.org/docs/user\\_guide/10min.html](https://pandas.pydata.org/docs/user_guide/10min.html)